Formatted: Font: Times New Roman, 14 pt

Infosys

Test Strategy for Continuous Validation
March, 2016

INFOSYS LIMITED

Bangalore

Document No.	<Bookstore_001>	Ver. Rev. :	1.0
Authorized by:	Prashant Burse	Signature/:	
		Date:	04-Apr-2016

COPYRIGHT NOTICE

©2016 Infosys Limited, Bangalore, India. All rights reserved. Infosys believes the information in this document is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. Except as expressly permitted, neither this document nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior permission of Infosys Limited and/or any named intellectual property rights holders under this document.

Infosys Limited
Hosur Road
Electronic City, 3rd Cross
Bangalore 560 100
India.
Telephone: (91) (80)28520 261-270
Fax: (91) (80) 8520 362
Website: http://www.infosys.com

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: (Default) Times New Roman, 14 pt

Author: Kaushal Rajvir, Bipulranjan Kumar

Date written (MM/DD/YY): 04/04/16

Project Details

- Project(s) involved: **IVS DevOps CoE**

Target readers: ALL

Keywords

DevOps, Continuous Testing, Continuous Testing in DevOps, Test Strategy, Continuous Test Strategy template

Table of Contents

1. Introduction .... 8~~7~~
1.1 Objective .... 8~~7~~
1.2 Continuous Validation .... 9~~7~~
1.3 Application Overview .... 12~~8~~
2. Continuous Validation Test Strategy .... 16~~9~~
2.1 Testing objective .... 16~~9~~
2.2 Test Approach .... 16~~9~~
2.3 Test Excellence – Adhering to testing standard .... 16~~9~~
2.3.1 Test Data Management .... 17~~9~~
2.3.2 Test Management .... 18~~10~~
2.3.3 SOA and Service Virtualization .... 21~~10~~

Formatted: Font: (Default) Times New Roman, Font color: Auto

Formatted: Font: (Default) Times New Roman, 14 pt, Check spelling and grammar

2.3.4 Test Automation 25~~10~~
2.3.5 Continuous Improvement through Feedback 28~~10~~
2.3.6 Source code Management 28~~10~~
2.4 Test types and Phases 30~~10~~
2.5 Setting up Automated Gating criteria 34~~11~~
2.6 Risk and Assumption 36~~12~~
3. Transition from old methodology to Continuous validation 39~~13~~
3.1 Objective 39~~13~~
3.2 Assess current capability 40~~13~~
3.3 Capability gaps, areas of improvement and priority 42~~13~~
3.4 Roadmap of milestone and execution timeline 44~~13~~
4. An illustrative list of activities of a continuous validation engineer 47~~14~~
5. Continuous Validation – Building Pipeline 48~~15~~
5.1 In Scope 48~~15~~
5.1.1 Test Data Setup 49~~15~~
5.1.2 Smoke Testing 51~~15~~
5.1.3 Functional Testing 53~~15~~
5.1.4 Middleware testing through Automation and Virtualization 54~~15~~
5.1.5 Performance Testing 56~~15~~
5.2 Tool Selection Strategy 57~~15~~
5.3 Test Automation Roadmap 59~~15~~
5.4 Test Environment Strategy 65~~16~~
5.4.1 Test Environment / Pre-production Environment 67~~16~~
5.4.2 Production Environment 69~~16~~
5.5 Test Execution and Test Automation Strategy 70~~16~~
5.6 Defect Management Strategy 73~~16~~
5.7 Setting up Automated Gating criteria 75~~16~~
5.8 Reporting & Test Metrics 76~~17~~
6. Reference(s) 79~~18~~

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Distribution List

Role	Name	Date	Signature

Formatted: Font: (Default) Times New Roman

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Revision History

Name	Date	Version	Changes
Kaushal Rajvir	29/3/2016	V0.1	Draft version
Kaushal Rajvir	31/3/2016	V0.2	Changes as per review comments by Prashant, Dinesh and Sameer
Kaushal Rajvir	04/04/2016	V0.3	Changes on Automation Roadmap, Daily activities suggested by Dinesh

Formatted: Font: (Default) Times New Roman

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

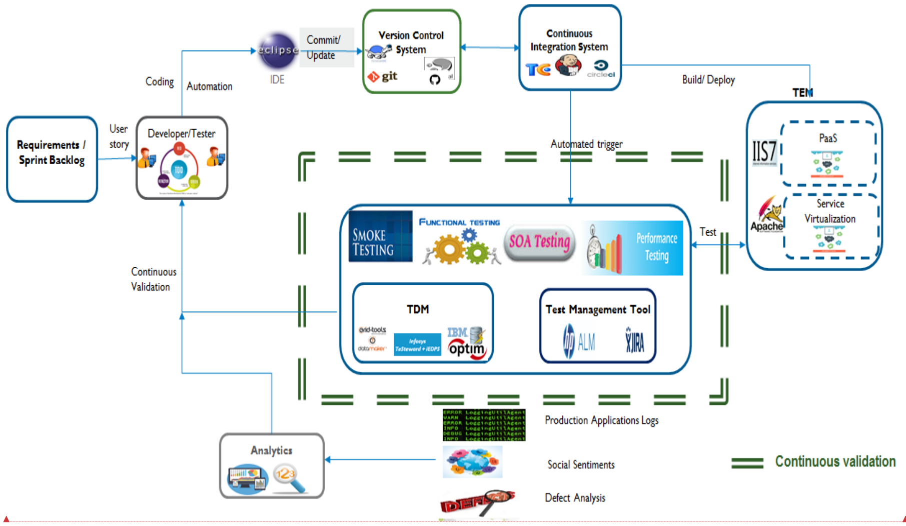
Formatted: Font: Times New Roman, 14 pt

~~7.~~1. Introduction

1.1 Objective

The current document discusses approach for continuous validation in the DevOps context for < *application name* > application. This is a living document that may be refined as the project progresses. The Project stakeholders - QA Manager, Test team lead, Project Manager and Develop manager would review and approve the final version of the document.

This continuous validation test strategy document will target the highlighted area within the entire DevOps lifecycle.



Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

1.2 Continuous Validation

Question ~~<~~ ***Reason for adopting continuous validation for the current application and intended benefits to be derived post implementation*** ~~>~~?

Ans:- Adopting continuous validation for your current application can bring several significant benefits. Here are some key reasons and the intended benefits post-implementation:

Reasons for Adopting Continuous Validation

1. **Improved Code Quality**:
   - Continuous validation ensures that code is constantly tested for functionality, security, and performance, leading to higher quality code[1].
2. **Early Detection of Issues**:
   - By validating code continuously, potential issues are identified and resolved early in the development cycle, reducing the risk of major defects[1].
3. **Enhanced Compliance**:
   - Continuous validation helps in maintaining compliance with industry standards and regulations by regularly checking code against these requirements[1].
4. **Increased Efficiency**:
   - Automating the validation process reduces the need for manual checks, allowing developers to focus on more value-added tasks[1].
5. **Better Alignment with Business Goals**:
   - Continuous validation ensures that the development efforts are aligned with business objectives by regularly reviewing and validating the strategy[1].

Intended Benefits Post-Implementation

1. **Reduced Deployment Risks**:
   - Continuous validation minimizes the risk of deploying faulty code, leading to more stable and reliable releases[1].
2. **Faster Time-to-Market**:
   - With automated validation, the development process becomes more efficient, enabling quicker releases and updates[1].
3. **Improved Customer Satisfaction**:
   - Higher quality and more reliable applications result in better user experiences and increased customer satisfaction[1].
4. **Cost Savings**:
   - Early detection and resolution of issues reduce the cost associated with fixing defects later in the development cycle[1].
5. **Scalability**:

Formatted: Font: Times New Roman, 14 pt, Bold, Font color: Text 1

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

  - Continuous validation supports scalable development practices, allowing the application to grow and adapt to new challenges and requirements[1].

Implementing continuous validation can significantly enhance the overall development process, leading to more robust and reliable applications.

Is there a specific aspect of continuous validation you're particularly interested in?

***Question** < Provide the details based on the current Gap analysis for continuous validation and high level overview of how it would be approached >*

Answer:- Gap Analysis for Continuous Validation

Current State:

1. ***Manual Testing:***
   - ***Testing is performed manually, leading to inconsistencies and potential human errors.***
2. ***Delayed Issue Detection:***
   - ***Issues are often detected late in the development cycle, increasing the cost and effort required to fix them.***
3. ***Lack of Automation:***
   - ***Limited or no automation in the validation process, resulting in slower feedback loops.***
4. ***Inconsistent Compliance:***
   - ***Difficulty in maintaining consistent compliance with industry standards and regulations.***
5. ***Resource Intensive:***
   - ***Manual validation is resource-intensive, requiring significant time and effort from the development team.***

Desired State:

1. ***Automated Testing:***
   - ***Implementing automated tests to ensure consistent and reliable validation.***
2. ***Early Issue Detection:***
   - ***Continuous validation to detect and resolve issues early in the development cycle.***
3. ***Enhanced Automation:***
   - ***Leveraging automation tools to streamline the validation process and provide faster feedback.***
4. ***Consistent Compliance:***
   - ***Regular automated checks to ensure compliance with industry standards and regulations.***
5. ***Efficient Resource Utilization:***
   - ***Reducing the manual effort required for validation, allowing the team to focus on more critical tasks.***

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Bold, Italic

Formatted: Font: Times New Roman, 14 pt, Bold, Italic

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt, Bold, Italic

High-Level Approach to Implement Continuous Validation

1. ***Assessment and Planning:***
    - ***Conduct a thorough assessment of the current validation processes.***
    - ***Identify key areas for improvement and define the scope of continuous validation.***
    - ***Develop a detailed implementation plan, including timelines and resource allocation.***
2. ***Tool Selection:***
    - ***Evaluate and select appropriate automation tools for continuous validation (e.g., Jenkins, Selenium, JUnit).***
    - ***Ensure the chosen tools integrate well with the existing development environment.***
3. ***Test Automation:***
    - ***Develop automated test scripts for various validation scenarios (unit tests, integration tests, performance tests).***
    - ***Implement continuous integration (CI) pipelines to run automated tests on every code commit.***
4. ***Compliance and Standards:***
    - ***Define and implement automated checks for compliance with industry standards and regulations.***
    - ***Regularly update the validation criteria to align with evolving standards.***
5. ***Monitoring and Reporting:***
    - ***Set up monitoring tools to track the results of continuous validation.***
    - ***Generate regular reports to provide insights into the validation process and identify areas for further improvement.***
6. ***Training and Support:***
    - ***Provide training to the development team on the new validation processes and tools.***
    - ***Establish a support system to address any issues or challenges encountered during implementation.***
7. ***Continuous Improvement:***
    - ***Regularly review and refine the continuous validation process to ensure it remains effective and efficient.***
    - ***Incorporate feedback from the development team to make necessary adjustments.***

Implementing continuous validation will help you achieve higher code quality, early issue detection, and more efficient resource utilization, ultimately leading to a more robust and reliable application.

Would you like more details on any specific step or tool mentioned?

Formatted: Font: Times New Roman, 14 pt

Formatted: Indent: Left: 0 cm

1.3 Application Overview

Question < ***Provide Application overview*** >

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answer:- Application Overview for DevOps of a New Project

Purpose

The purpose of this overview is to outline the key aspects of a new project from a DevOps perspective, ensuring efficient development, deployment, and maintenance processes.

Key Features

1. **Admin Interface:**
    - **User Management: Admins can manage user roles and permissions.**
    - **Content Management: Admins can add, edit, and delete content.**
2. **User Interface:**
    - **Responsive Design: Ensures the application is accessible on various devices.**
    - **User-Friendly Navigation: Intuitive navigation for a seamless user experience.**
3. **Data Persistence:**
    - **Database Integration: Use of a robust database system for data storage and retrieval.**
    - **Backup and Recovery: Regular backups and recovery plans to ensure data integrity.**

Technical Overview

1. **Frontend:**

  - Frameworks: Use modern frameworks like React or Angular for building the user interface.
  - Styling: Use CSS frameworks like Bootstrap or Tailwind CSS for responsive design.

2. Backend:
   - API Development: Develop RESTful APIs using frameworks like Spring Boot or Express.js.
   - Database: Use relational databases like PostgreSQL or MySQL, or NoSQL databases like MongoDB.

3. Automation and CI/CD:
   - Continuous Integration (CI): Automated builds and tests using tools like Jenkins or GitHub Actions.
   - Continuous Deployment (CD): Automated deployment processes using Docker and Kubernetes.

DevOps Considerations

1. Infrastructure as Code (IaC):
   - Use tools like Terraform or Ansible to manage infrastructure setup and configuration.

2. CI/CD Pipeline:
   - Version Control: Use Git for source code management.
   - Build Automation: Use Jenkins or GitHub Actions for automated builds.
   - Testing: Implement automated testing (unit, integration, and performance tests) using tools like JUnit and Selenium.
   - Deployment: Use Docker for containerization and Kubernetes for orchestration.

3. Monitoring and Logging:

  - Implement monitoring tools like Prometheus and Grafana to track application performance.
  - Use logging tools like ELK Stack (Elasticsearch, Logstash, Kibana) for centralized logging and analysis.

4. Security:
   - Implement security best practices, including automated security scans and vulnerability assessments.
   - Use tools like OWASP ZAP for security testing.
5. Scalability and Reliability:
   - Design the application to be scalable, using load balancers and auto-scaling groups.
   - Ensure high availability with redundant systems and failover mechanisms.

Benefits for DevOps

- Automated Processes: Reduced manual intervention with automated builds, tests, and deployments.
- Early Issue Detection: Continuous validation and testing to catch issues early in the development cycle.
- Efficient Resource Utilization: Optimized use of resources through automation and monitoring.
- Scalability: Easily scalable infrastructure to handle growing demands.
- Enhanced Security: Regular security checks and automated vulnerability assessments.

Implementing these DevOps practices will ensure a robust, scalable, and secure application, providing a seamless experience for both admins and users.

Would you like more details on any specific DevOps tool or practice?

Formatted: Font: Times New Roman, 14 pt

~~8.~~2. Continuous Validation Test Strategy

2.1 Testing objective

- To test and prove that the requirements, features and code are fit to move into the next level.
- System, Acceptance testing will provide the information as to whether the features are ready for the production.
- Mitigate the product risks, highlight issues and new risks.
- Finding defects sooner through continuous testing
- Faster feedback through automated testing

2.2 Test Approach

To successfully test and deliver the product in continuous validation environment.

- Test as early as possible –
- Test as often as possible -
    - Continuously validate with every deployment or code upgrade
    - Adapt and Evolve with each delivery and test cycle
- Test maximum & Prioritize
- Test most important features first
- The features/requirements that are planned to be delivered for iteration will be reviewed for product risk and an appropriate testing approach will be developed for each iteration.
- Automation Regression, Build Verification testing
    - 100% Automated Execution
    - Execute automation scripts for every deployment

Test cases development will base on the documentation (Requirements, Story boards) available. The key difference is test case prioritization, as discussed above it be carried out after every scrum meeting.

2.3 Test Excellence – Adhering to testing standard

As a part of the mature testing delivery practices, the following standards will be followed during the continuous validation.

2.3.1 Test Data Management

< Provide the automated test data set up strategy for the current application, identify the TD needs – ex: whether mockup is required, how frequently refresh is required, what are the test data manipulation needs – ex: add, modify or delete etc. >

Answer:-Automated Test Data Setup Strategy for New DevOps Project

Test Data Needs

1. **Mockup Data:**
   - **Requirement: Yes, mockup data is essential to simulate real-world scenarios for testing.**
   - **Types: User data, transaction records, configuration settings, and other relevant data specific to the application.**
2. **Data Refresh Frequency:**
   - **Frequency: Test data should be refreshed before each major testing cycle (e.g., before integration testing, system testing).**
   - **Automated Refresh: Implement automated scripts to refresh test data at scheduled intervals or on-demand.**
3. **Test Data Manipulation Needs:**
   - **Add: Ability to add new test data entries to test new features or scenarios.**
   - **Modify: Ability to modify existing test data to test different conditions and edge cases.**
   - **Delete: Ability to delete test data to ensure proper handling of data removal and cleanup.**

Automated Test Data Setup Strategy

1. **Data Generation:**
   - **Automated Scripts: Develop scripts to generate mockup data, ensuring it covers a wide range of scenarios.**
   - **Data Variability: Ensure the generated data includes various edge cases and typical use cases.**
2. **Data Storage:**
   - **Database: Use a separate test database to store the generated test data.**
   - **File System: Store files and other non-database data in a designated test directory.**
3. **Data Refresh:**
   - **Scheduled Jobs: Set up scheduled jobs to refresh test data at specified intervals.**

  - On-Demand Refresh: Provide the ability to trigger data refresh manually when needed.

4. **Data Manipulation:**
   - **APIs for Test Data Management: Develop APIs to add, modify, and delete test data entries.**
   - **User Interface: Provide a simple UI for testers to manage test data if needed.**
5. **Integration with CI/CD Pipeline:**
   - **Pre-Test Setup: Include test data setup scripts in the CI/CD pipeline to ensure the test environment is prepared before running tests.**
   - **Post-Test Cleanup: Implement cleanup scripts to remove test data after testing is complete.**
6. **Version Control:**
   - **Test Data Scripts: Store test data generation and manipulation scripts in version control (e.g., Git) to track changes and ensure consistency.**
7. **Monitoring and Logging:**
   - **Logs: Implement logging for test data setup and manipulation activities to track any issues.**
   - **Monitoring: Monitor the test data environment to ensure data integrity and availability.**

By implementing this automated test data setup strategy, you can ensure that your new DevOps project is thoroughly tested with realistic data, leading to higher quality and more reliable releases.

Would you like more details on any specific aspect of this strategy?

Formatted: Font: Times New Roman, 14 pt

2.3.2 Test Management

*Question< **Provide the test management tool details, standards for test cases design, automated test cases execution status update and defects flow etc for the current application** >*

Formatted: Font: Times New Roman, 14 pt, Italic

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt, Italic

Answer:- Test Management Tool Details

Recommended Test Management Tools

1. *PractiTest:*
    - *Features: End-to-end SaaS test management platform, integrates with various CI/CD tools, supports agile and waterfall methodologies.*
    - *Benefits: Centralized management of test cases, real-time reporting, and seamless integration with bug tracking tools*[1]*.*
2. *TestRail:*
    - *Features: Comprehensive test case management, customizable dashboards, and detailed reporting.*
    - *Benefits: Easy integration with CI/CD pipelines, supports manual and automated testing, and provides robust analytics*[1]*.*
3. *Tricentis qTest:*
    - *Features: Supports agile and DevOps workflows, integrates with Jira, and provides real-time insights.*
    - *Benefits: Scalable for large teams, enhances collaboration, and ensures traceability*[1]*.*

Standards for Test Case Design

1. *Clarity and Conciseness:*
    - *Test cases should be clear, concise, and easy to understand.*
    - *Use simple language and avoid ambiguity.*
2. *Traceability:*
    - *Each test case should be traceable to specific requirements or user stories.*
    - *Include references to requirement IDs or user story IDs.*
3. *Reusability:*
    - *Design test cases to be reusable across different test scenarios.*
    - *Modularize test steps to avoid duplication.*

4. ***Coverage:***
    - ***Ensure comprehensive coverage of all functional and non-functional requirements.***
    - ***Include positive, negative, and edge case scenarios.***
5. ***Maintainability:***
    - ***Test cases should be easy to maintain and update.***
    - ***Use consistent naming conventions and document any dependencies.***

Automated Test Cases Execution Status Update

1. ***Integration with CI/CD Pipeline:***
    - ***Integrate automated test cases with the CI/CD pipeline using tools like Jenkins or GitHub Actions.***
    - ***Ensure test cases are executed automatically on code commits or scheduled intervals.***
2. ***Real-Time Reporting:***
    - ***Use test management tools to provide real-time updates on test execution status.***
    - ***Generate automated reports and dashboards to track test progress and results.***
3. ***Notifications:***
    - ***Set up notifications for test execution status updates.***
    - ***Notify relevant stakeholders of test results, failures, and any critical issues.***

Defects Flow

1. ***Defect Reporting:***
    - ***Report defects using the integrated bug tracking tool (e.g., Jira, Bugzilla).***
    - ***Include detailed information such as steps to reproduce, expected vs. actual results, and severity.***
2. ***Defect Triage:***

- *Conduct regular defect triage meetings to prioritize and assign defects.*
- *Ensure defects are categorized based on severity and impact.*

3. *Defect Resolution:*
    - *Assign defects to the appropriate developers for resolution.*
    - *Track the progress of defect resolution and ensure timely fixes.*
4. *Defect Verification:*
    - *Verify fixed defects through retesting.*
    - *Update the defect status to "Closed" once the fix is confirmed.*
5. *Defect Metrics and Reporting:*
    - *Track defect metrics such as defect density, resolution time, and reopen rate.*
    - *Generate reports to analyze defect trends and identify areas for improvement.*

Implementing these standards and practices will help ensure efficient test management, accurate test execution status updates, and effective defect handling for your current application.

Would you like more details on any specific tool or process?

Formatted: Font: Times New Roman, 14 pt

2.3.3 SOA and Service Virtualization

Question< *Identify SOA testing needs for an API based application. Provide the details around why service virtualization and which all services have been virtualized, tools used for service virtualization etc. for the current application.* >

Formatted: Font: Times New Roman, 14 pt

Answer:- SOA Testing Needs for an API-Based Application

SOA Testing Needs

1. Functional Testing:
    - Ensure that each service performs its intended function correctly.

  - Validate the integration between different services.

2. **Performance Testing:**
   - Assess the performance of services under various load conditions.
   - Identify bottlenecks and optimize service performance.

3. **Security Testing:**
   - Verify that services are secure and protected against vulnerabilities.
   - Ensure compliance with security standards and regulations.

4. **Interoperability Testing:**
   - Ensure that services can interact seamlessly with other services and systems.
   - Validate data exchange and communication protocols.

5. **Regression Testing:**
   - Ensure that new changes do not negatively impact existing functionality.
   - Automate regression tests to maintain consistency.

Why Service Virtualization?

1. **Dependency Management:**
   - Service virtualization allows testing of services even when dependent components are unavailable or incomplete[1].
   - It simulates the behavior of dependent services, enabling continuous testing without waiting for all components to be ready[1].

2. **Cost Efficiency:**
   - Reduces the need for setting up and maintaining complex test environments.
   - Saves costs associated with using third-party services for testing[1].

3. **Parallel Development:**

   - **Enables parallel development and testing by providing virtual services that mimic real ones[1].**
   - **Facilitates early detection of issues, reducing the overall development cycle time[1].**

4. **Consistent Testing Environment:**
   - **Provides a stable and consistent testing environment, reducing flakiness in test results[1].**
   - **Ensures reliable and repeatable test outcomes.**

Services Virtualized

1. **Payment Gateway:**
   - **Simulates interactions with external payment processing services.**
   - **Allows testing of various payment scenarios without actual transactions.**
2. **User Authentication:**
   - **Virtualizes authentication services to test login, logout, and session management.**
   - **Ensures security and functionality without relying on live authentication servers.**
3. **Third-Party APIs:**
   - **Simulates responses from third-party APIs (e.g., weather services, social media integrations).**
   - **Enables testing of API interactions without depending on external services.**
4. **Database Services:**
   - **Virtualizes database interactions to test CRUD operations.**
   - **Ensures data integrity and performance without using a live database.**

Tools Used for Service Virtualization

1. **Parasoft Virtualize:**

   - Provides comprehensive service virtualization capabilities.
   - Supports a wide range of protocols and integrates with CI/CD pipelines[2].

2. WireMock:
   - An open-source tool for mocking HTTP services.
   - Easy to set up and use for simulating RESTful APIs[2].

3. Mountebank:
   - An open-source tool that supports multiple protocols (HTTP, HTTPS, TCP).
   - Allows for complex service virtualization scenarios[2].

4. SmartBear ServiceV Pro:
   - Offers advanced service virtualization features.
   - Integrates with other SmartBear tools for end-to-end testing[2].

5. CA Service Virtualization (Broadcom):
   - Provides robust service virtualization for complex enterprise environments.
   - Supports a wide range of protocols and use cases[2].

Implementing service virtualization in your SOA testing strategy will enhance your ability to test services independently, reduce dependencies, and improve overall testing efficiency.

Would you like more details on any specific tool or aspect of service virtualization?

Formatted: Font: Times New Roman, 14 pt

2.3.4 Test Automation

Question<*Provide the details around the framework used for the automation (BDD, TDD, Hybrid etc.), tools for automation (QTP, Selenium, fitnesse, jBehave etc.), test cases have been automated etc. for the current application* >

Formatted: Font: Times New Roman, 14 pt

Answer:- SOA Testing Needs for an API-Based Application

SOA Testing Needs

1. **Functional Testing:**
    - **Ensure that each service performs its intended function correctly.**
    - **Validate the integration between different services.**
2. **Performance Testing:**
    - **Assess the performance of services under various load conditions.**
    - **Identify bottlenecks and optimize service performance.**
3. **Security Testing:**
    - **Verify that services are secure and protected against vulnerabilities.**
    - **Ensure compliance with security standards and regulations.**
4. **Interoperability Testing:**
    - **Ensure that services can interact seamlessly with other services and systems.**
    - **Validate data exchange and communication protocols.**
5. **Regression Testing:**
    - **Ensure that new changes do not negatively impact existing functionality.**
    - **Automate regression tests to maintain consistency.**

Why Service Virtualization?

1. **Dependency Management:**

  - Service virtualization allows testing of services even when dependent components are unavailable or incomplete[1].
  - It simulates the behavior of dependent services, enabling continuous testing without waiting for all components to be ready[1].

2. Cost Efficiency:
  - Reduces the need for setting up and maintaining complex test environments.
  - Saves costs associated with using third-party services for testing[1].

3. Parallel Development:
  - Enables parallel development and testing by providing virtual services that mimic real ones[1].
  - Facilitates early detection of issues, reducing the overall development cycle time[1].

4. Consistent Testing Environment:
  - Provides a stable and consistent testing environment, reducing flakiness in test results[1].
  - Ensures reliable and repeatable test outcomes.

Services Virtualized

1. Payment Gateway:
  - Simulates interactions with external payment processing services.
  - Allows testing of various payment scenarios without actual transactions.

2. User Authentication:
  - Virtualizes authentication services to test login, logout, and session management.
  - Ensures security and functionality without relying on live authentication servers.

3. Third-Party APIs:

  - Simulates responses from third-party APIs (e.g., weather services, social media integrations).
  - Enables testing of API interactions without depending on external services.

4. Database Services:
   - Virtualizes database interactions to test CRUD operations.
   - Ensures data integrity and performance without using a live database.

Tools Used for Service Virtualization

1. Parasoft Virtualize:
   - Provides comprehensive service virtualization capabilities.
   - Supports a wide range of protocols and integrates with CI/CD pipelines[2].
2. WireMock:
   - An open-source tool for mocking HTTP services.
   - Easy to set up and use for simulating RESTful APIs[2].
3. Mountebank:
   - An open-source tool that supports multiple protocols (HTTP, HTTPS, TCP).
   - Allows for complex service virtualization scenarios[2].
4. SmartBear ServiceV Pro:
   - Offers advanced service virtualization features.
   - Integrates with other SmartBear tools for end-to-end testing[2].
5. CA Service Virtualization (Broadcom):
   - Provides robust service virtualization for complex enterprise environments.
   - Supports a wide range of protocols and use cases[2].

Implementing service virtualization in your SOA testing strategy will enhance your ability to test services independently, reduce dependencies, and improve overall testing efficiency.

Would you like more details on any specific tool or aspect of service virtualization?

Formatted: Font: (Default) Times New Roman, 14 pt

2.3.5 Continuous Improvement through Feedback

Under the continuous improvement, any defect or deviation from the expected result will automatically be logged in the defect management tool specified for the delivery and notification to be sent to the defect owner and project manager.

As a part of continuous improvement and feedback mechanism, there will also be required to conduct analysis on the defects found for the application and an improvement plan has to be prepared at end of each sprint or pre-decided interval.

2.3.6 Source code Management

Question< *If there are any repository tool being used / managed through Dev libraries, then please provide the reference link along with the path which specifies how source code versioning is managed across the delivery. Or provide the details around how the feature files, Dev libraries, QA automation script library will be managed under project folders. Also provide the details around how the version controlling will be achieved for all source code (feature files, QA automation scripts, libraries etc.).* >

Formatted: Font: Times New Roman, 14 pt

Answer:- **Repository Tool and Source Code Versioning**
Repository Tool
For managing source code versioning, a popular and widely used tool is **Git**. Git is a distributed version control system that allows multiple developers to work on the same project simultaneously. Here are some key tools and platforms that utilize Git:

1. **GitHub**:
    - **URL**: GitHub
    - **Features**: Provides a web-based interface for Git repositories, supports collaboration, pull requests, and issue tracking.
2. **GitLab**:
    - **URL**: GitLab
    - **Features**: Offers integrated CI/CD pipelines, project management tools, and robust security features.
3. **Bitbucket**:

  - **URL**: Bitbucket
  - **Features**: Supports Git and Mercurial repositories, integrates with Jira for project management, and provides CI/CD capabilities.

Project Folder Structure and Management

1. **Feature Files**:
   - **Path**: src/main/features/
   - **Description**: Contains all the feature files, organized by functionality or module.
   - **Version Control**: Managed using Git, with each feature file tracked for changes and updates.
2. **Dev Libraries**:
   - **Path**: src/main/libs/
   - **Description**: Contains custom libraries and dependencies used by the application.
   - **Version Control**: Libraries are versioned to ensure compatibility and traceability.
3. **QA Automation Script Library**:
   - **Path**: src/test/automation/
   - **Description**: Contains automated test scripts for functional, integration, and performance testing.
   - **Version Control**: Scripts are versioned to maintain consistency and track changes over time.

Version Controlling Strategy

1. **Branching Strategy**:
   - **Main Branch**: The main branch (e.g., main or master) contains the stable version of the code.
   - **Feature Branches**: Each new feature or bug fix is developed in a separate branch (e.g., feature/feature-name).
   - **Merge Requests**: Changes are merged into the main branch through pull requests or merge requests, ensuring code reviews and approvals.
2. **Commit Messages**:
   - **Standards**: Use clear and descriptive commit messages to explain the changes made.
   - **Format**: Follow a consistent format, such as type(scope): description (e.g., feat(login): add user authentication).
3. **Continuous Integration (CI)**:
   - **Tools**: Use CI tools like Jenkins, GitHub Actions, or GitLab CI to automate the build and testing process.
   - **Pipeline**: Set up pipelines to run automated tests, build the application, and deploy to staging or production environments.
4. **Tagging and Releases**:
   - **Tags**: Use tags to mark specific versions or releases (e.g., v1.0.0).
   - **Releases**: Create release notes to document changes, new features, and bug fixes for each version.

5. **Code Reviews**:
    - **Process**: Implement a code review process to ensure code quality and adherence to standards.
    - **Tools**: Use tools like GitHub Pull Requests or GitLab Merge Requests for code reviews.

By following this structured approach, you can effectively manage and version control all source code, feature files, QA automation scripts, and libraries, ensuring a smooth and efficient development process.

Would you like more details on any specific aspect of this strategy?

Formatted: Font: Times New Roman, 14 pt

2.4 Test types and Phases

Following is the list of various testing that needs to be carried out as part of the testing along with the objective of each test and the characteristics

- Validation of Acceptance
    - Build verification or Smoke Test
- Validation of Tasks implementing the User Stories
    - Functional Testing
    - Regression testing <*Provide the details of how regression test cases are identified, frequency of the execution is planned etc* >

      Answer:- **Identifying Regression Test Cases**

      Steps to Identify Regression Test Cases

1. **Analyze Recent Changes**:
    - Identify areas of the application that have been modified, including new features, bug fixes, and enhancements.
    - Focus on modules and components that are directly impacted by these changes.
2. **Review Critical Functionalities**:
    - Identify core functionalities that are critical to the application's operation.
    - Ensure these functionalities are included in the regression test suite to prevent any disruptions.
3. **Assess High-Risk Areas**:
    - Determine areas of the application that are prone to defects or have a history of frequent issues.
    - Prioritize these areas for regression testing to mitigate potential risks.
4. **Leverage Test Coverage Reports**:
    - Use test coverage tools to identify gaps in existing test cases.

  - Ensure comprehensive coverage of all impacted areas by adding or updating test cases as needed.

5. **Consult with Stakeholders**:
   - Collaborate with developers, QA engineers, and product owners to identify critical test cases.
   - Incorporate feedback from stakeholders to ensure all important scenarios are covered.

Planning Frequency of Regression Test Execution

Factors to Consider

1. **Release Cycle**:
   - Align regression testing frequency with the application's release cycle.
   - For frequent releases, consider running regression tests before each release.
2. **Development Methodology**:
   - In Agile environments, run regression tests at the end of each sprint or iteration.
   - For continuous delivery, integrate regression tests into the CI/CD pipeline for regular execution.
3. **Change Impact**:
   - Assess the impact of recent changes on the application.
   - For significant changes, run a full regression test suite; for minor changes, run targeted regression tests.
4. **Resource Availability**:
   - Consider the availability of testing resources, including time, tools, and personnel.
   - Optimize the regression test suite to balance thoroughness with efficiency.

Recommended Frequency

1. **Daily**:
   - For continuous integration and delivery environments, run automated regression tests daily.
   - Ensure quick feedback on code changes and maintain application stability.
2. **End of Sprint**:
   - In Agile development, run regression tests at the end of each sprint.
   - Validate that new features and changes have not introduced any regressions.
3. **Pre-Release**:
   - Run a comprehensive regression test suite before each major release.
   - Ensure all critical functionalities are working as expected and no new defects have been introduced.

Tools and Best Practices

Tools for Regression Testing

1. **Selenium**:

   - Automates web application testing and supports multiple browsers and platforms.
   - Ideal for functional and regression testing of web applications.
2. **JUnit**:
   - A widely used testing framework for Java applications.
   - Supports unit and regression testing with easy integration into CI/CD pipelines.
3. **TestNG**:
   - A testing framework inspired by JUnit, with additional features for parallel test execution and configuration.
   - Suitable for both unit and regression testing.
4. **Jenkins**:
   - An open-source automation server that supports CI/CD pipelines.
   - Integrates with various testing tools to automate regression test execution.

Best Practices

1. **Automate Regression Tests**:
   - Automate as many regression tests as possible to ensure consistency and efficiency.
   - Use tools like Selenium, JUnit, and TestNG for automation.
2. **Maintain Test Suites**:
   - Regularly update and maintain regression test suites to reflect changes in the application.
   - Remove obsolete test cases and add new ones as needed.
3. **Prioritize Test Cases**:
   - Prioritize test cases based on criticality and risk.
   - Focus on high-impact areas to ensure thorough validation.
4. **Monitor Test Results**:
   - Continuously monitor and analyze test results to identify trends and areas for improvement.
   - Use reporting tools to generate insights and track test coverage.

By following these strategies and best practices, you can effectively identify and manage regression test cases, ensuring the stability and reliability of your application.

- Middleware Testing through Automation or Virtualization
- Performance Testing
- **<Provide any other testing phases to be used>**

Answer:- **Additional Testing Phases**

In addition to the standard phases of testing, there are several other important testing phases that can be used to ensure comprehensive coverage and quality assurance for your application. Here are some key additional testing phases:

Formatted: Indent: Left: 3.17 cm, No bullets or numbering

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

1. **Unit Testing**:
    - **Purpose**: To test individual components or units of the application in isolation.
    - **Focus**: Ensures that each unit functions correctly.
    - **Tools**: JUnit, NUnit, TestNG.
2. **Integration Testing**:
    - **Purpose**: To test the interaction between integrated units or components.
    - **Focus**: Ensures that combined components work together as expected.
    - **Tools**: JUnit, TestNG, Postman for API integration.
3. **System Testing**:
    - **Purpose**: To test the complete and integrated application as a whole.
    - **Focus**: Validates the end-to-end functionality of the application.
    - **Tools**: Selenium, QTP, TestComplete.
4. **Acceptance Testing**:
    - **Purpose**: To verify that the application meets the business requirements and is ready for deployment.
    - **Focus**: Ensures that the application is acceptable to the end-users.
    - **Tools**: Cucumber, FitNesse.
5. **Performance Testing**:
    - **Purpose**: To assess the application’s performance under various conditions.
    - **Focus**: Evaluates response times, throughput, and resource usage.
    - **Tools**: JMeter, LoadRunner, Gatling.
6. **Security Testing**:
    - **Purpose**: To identify vulnerabilities and ensure the application is secure.
    - **Focus**: Protects against threats and ensures data integrity.
    - **Tools**: OWASP ZAP, Burp Suite, Nessus.
7. **Usability Testing**:
    - **Purpose**: To evaluate the application’s user interface and user experience.
    - **Focus**: Ensures the application is user-friendly and intuitive.
    - **Tools**: UserTesting, Crazy Egg, Hotjar.
8. **Compatibility Testing**:
    - **Purpose**: To ensure the application works across different devices, browsers, and operating systems.
    - **Focus**: Validates compatibility with various environments.
    - **Tools**: BrowserStack, Sauce Labs, CrossBrowserTesting.
9. **Regression Testing**:
    - **Purpose**: To verify that recent changes have not adversely affected existing functionality.

  - **Focus**: Ensures stability and reliability after updates.
  - **Tools**: Selenium, TestNG, Jenkins for automation.

10. **Exploratory Testing**:
    - **Purpose**: To explore the application without predefined test cases.
    - **Focus**: Identifies unexpected issues and improves overall quality.
    - **Tools**: Session-based test management (SBTM) tools.
11. **Alpha and Beta Testing**:
    - **Alpha Testing**: Conducted by internal teams to identify bugs before releasing to external users.
    - **Beta Testing**: Conducted by a limited group of external users to gather feedback and identify issues in a real-world environment.

By incorporating these additional testing phases, you can ensure a thorough and comprehensive testing process, leading to a higher quality and more reliable application.

Would you like more details on any specific testing phase or tool?

Formatted: Font: Times New Roman, 14 pt

2.5 Setting up Automated Gating criteria

Gating will be implemented for the following –

- Code Quality Gate
- Unit Test Gate
- Functional Smoke Test Gate
- Functional Testing Gate
- SOA Testing Gate
- Performance Smoke Testing Gate
- Load Testing Gate
- Stress Testing Gate
- Endurance Testing Gate
- Deployment Gate

Automated Gating will be implemented (expect PT) and the following type of gating shall be implemented for all phases –

Gating Type	Gating Criteria
Defects Gating	Count of Critical / High Severity Code Quality Issues greater than

Formatted: Font: (Default) Times New Roman, 14 pt

	threshold value - Do not proceed with Next step
Pass% Gating	Test Cases Pass % less than threshold value (70% for most gating)

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

2.6 Risk and Assumption

< Provide the details of risk and assumption for the current application in the context of continuous validation >

Answer:- **Risks and Assumptions for Continuous Validation in the Current Application**

Risks

1. **Incomplete Test Coverage**:
   - **Risk**: Not all scenarios and edge cases are covered by automated tests.
   - **Mitigation**: Regularly review and update test cases to ensure comprehensive coverage.
2. **False Positives/Negatives**:
   - **Risk**: Automated tests may produce false positives (indicating issues where there are none) or false negatives (failing to detect actual issues).
   - **Mitigation**: Continuously refine test scripts and validation criteria to improve accuracy.
3. **Tool Integration Issues**:
   - **Risk**: Integration of continuous validation tools with existing CI/CD pipelines may face compatibility issues.
   - **Mitigation**: Conduct thorough compatibility testing and use well-documented tools with strong community support.
4. **Performance Overhead**:
   - **Risk**: Continuous validation processes may introduce performance overhead, slowing down the development pipeline.
   - **Mitigation**: Optimize validation scripts and run performance-intensive tests during off-peak hours.
5. **Security Vulnerabilities**:
   - **Risk**: Automated validation processes may inadvertently expose security vulnerabilities.
   - **Mitigation**: Implement security best practices and conduct regular security audits.
6. **Data Integrity Issues**:
   - **Risk**: Continuous validation may lead to data corruption or loss if not properly managed.
   - **Mitigation**: Use robust data management practices and backup strategies.

Assumptions

1. **Stable Test Environment**:
   - **Assumption**: The test environment is stable and mirrors the production environment closely.
   - **Impact**: Ensures that test results are reliable and reflective of real-world scenarios.

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

2. **Availability of Resources**:
   - **Assumption**: Sufficient resources (e.g., hardware, software, personnel) are available to support continuous validation.
   - **Impact**: Enables smooth execution of validation processes without resource constraints.
3. **Tool Compatibility**:
   - **Assumption**: The chosen validation tools are compatible with the existing technology stack.
   - **Impact**: Facilitates seamless integration and reduces the risk of tool-related issues.
4. **Regular Updates**:
   - **Assumption**: Test cases and validation scripts are regularly updated to reflect changes in the application.
   - **Impact**: Ensures that validation processes remain relevant and effective.
5. **Stakeholder Buy-In**:
   - **Assumption**: All stakeholders (developers, QA, management) are committed to the continuous validation process.
   - **Impact**: Promotes collaboration and ensures that validation efforts are supported and prioritized.
6. **Automated Processes**:
   - **Assumption**: Most validation processes are automated to reduce manual intervention and increase efficiency.
   - **Impact**: Enhances the speed and reliability of validation efforts.

By understanding and addressing these risks and assumptions, you can ensure a more effective and reliable continuous validation process for your application.

Would you like more details on any specific risk or assumption?

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

~~9.~~3. Transition from old methodology to Continuous validation

The approach to transition from old methodology to or up to DevOps - continuous validation is extreme critical factor for a successful transition.

The traditional planned approach to organization change management is replaced by an adaptive, strategy based roll out of continuous delivery efforts. The following approach has been adopted for the <*application name*> transition.

3.1 Objective

< Provide the details of why transition from old methodology to DevOps / Continuous validation is required. List down key objective / performance improvement parameters which are commonly found during each deployment. This should be found out during gap analysis.>

Answer:- **Transition from Old Methodology to DevOps/Continuous Validation**

Why Transition is Required

1. **Improved Code Quality**:
    - Continuous validation ensures that code is constantly tested for functionality, security, and performance, leading to higher quality code[1].
2. **Early Detection of Issues**:
    - By validating code continuously, potential issues are identified and resolved early in the development cycle, reducing the risk of major defects[1].
3. **Enhanced Compliance**:
    - Continuous validation helps in maintaining compliance with industry standards and regulations by regularly checking code against these requirements[1].
4. **Increased Efficiency**:
    - Automating the validation process reduces the need for manual checks, allowing developers to focus on more value-added tasks[1].
5. **Better Alignment with Business Goals**:
    - Continuous validation ensures that the development efforts are aligned with business objectives by regularly reviewing and validating the strategy[1].

Key Objectives/Performance Improvement Parameters

1. **Reduced Deployment Risks**:
    - Continuous validation minimizes the risk of deploying faulty code, leading to more stable and reliable releases[1].
2. **Faster Time-to-Market**:
    - With automated validation, the development process becomes more efficient, enabling quicker releases and updates[1].
3. **Improved Customer Satisfaction**:

  - Higher quality and more reliable applications result in better user experiences and increased customer satisfaction[1].
4. **Cost Savings**:
  - Early detection and resolution of issues reduce the cost associated with fixing defects later in the development cycle[1].
5. **Scalability**:
  - Continuous validation supports scalable development practices, allowing the application to grow and adapt to new challenges and requirements[1].

By transitioning to DevOps and continuous validation, your application can achieve these objectives, leading to a more efficient and reliable development process.

Would you like more details on any specific aspect of this transition? [1]: 1

Formatted: Font: Times New Roman, 14 pt

3.2 Assess current capability

< Provide the details of the current capability of the delivery process, key skills of the DevOps – Continuous validation team, Internal Continuous validation expert etc. >

Answer:- **Current Capability of the Delivery Process**

Current Capability

1. **Automated CI/CD Pipeline**:
  - **Integration**: Continuous Integration (CI) tools like Jenkins, GitHub Actions, or GitLab CI are used to automate the build and testing process.
  - **Deployment**: Continuous Deployment (CD) ensures that code changes are automatically deployed to staging or production environments after passing all tests[1].
2. **Comprehensive Testing**:
  - **Unit Testing**: Automated unit tests validate individual components.
  - **Integration Testing**: Ensures that integrated components work together as expected.
  - **System Testing**: Validates the entire system's functionality.
  - **Performance Testing**: Assesses the application's performance under various conditions[1].
3. **Monitoring and Logging**:
  - **Real-Time Monitoring**: Tools like Prometheus and Grafana are used to monitor application performance.
  - **Centralized Logging**: ELK Stack (Elasticsearch, Logstash, Kibana) is used for centralized logging and analysis[1].
4. **Security**:
  - **Automated Security Scans**: Tools like OWASP ZAP and Burp Suite are used for regular security assessments.
  - **Compliance**: Continuous validation ensures compliance with industry standards and regulations[1].

Key Skills of the DevOps – Continuous Validation Team

1. **Automation Expertise**:
    - **Scripting**: Proficiency in scripting languages like Python, Bash, or PowerShell for automation tasks.
    - **CI/CD Tools**: Experience with CI/CD tools like Jenkins, GitHub Actions, GitLab CI, and CircleCI[2].
2. **Testing Skills**:
    - **Automated Testing**: Knowledge of automated testing frameworks like Selenium, JUnit, TestNG, and Cypress.
    - **Performance Testing**: Skills in tools like JMeter and LoadRunner for performance testing[2].
3. **Infrastructure Management**:
    - **Infrastructure as Code (IaC)**: Experience with tools like Terraform, Ansible, and CloudFormation for managing infrastructure.
    - **Containerization**: Proficiency in Docker and Kubernetes for containerization and orchestration[2].
4. **Monitoring and Logging**:
    - **Monitoring Tools**: Experience with Prometheus, Grafana, and Nagios for monitoring.
    - **Logging Tools**: Skills in using ELK Stack for centralized logging[2].
5. **Security**:
    - **Security Testing**: Knowledge of security testing tools like OWASP ZAP and Burp Suite.
    - **Compliance**: Understanding of security standards and compliance requirements[2].

Internal Continuous Validation Expert

1. **Role and Responsibilities**:
    - **Strategy Development**: Develop and implement continuous validation strategies to ensure code quality and compliance.
    - **Tool Integration**: Integrate continuous validation tools into the CI/CD pipeline.
    - **Training and Mentorship**: Provide training and mentorship to the team on best practices for continuous validation[3].
2. **Skills and Expertise**:
    - **Deep Understanding of CI/CD**: Expertise in continuous integration and continuous deployment practices.
    - **Automation Proficiency**: Strong skills in automation tools and scripting languages.
    - **Testing Knowledge**: Extensive experience in automated testing frameworks and performance testing tools.
    - **Security Awareness**: Knowledge of security testing and compliance standards[3].

By leveraging these capabilities and skills, the DevOps – Continuous Validation team can ensure a robust, efficient, and secure delivery process for your application.

Would you like more details on any specific aspect of this transition? [1]: 10 [2]: 4 [3]: 2

Formatted: Font: Times New Roman, 14 pt

3.3 Capability gaps, areas of improvement and priority

< Identify and provide the capability gap, improvement scope to bridge the capability gap and priority order to achieve them. >

Answer:- **Capability Gap Analysis for Continuous Validation Transition**

Current Capability Gaps

1. **Manual Testing**:
   - **Gap**: Testing is performed manually, leading to inconsistencies and potential human errors.
   - **Impact**: Slower feedback loops and higher chances of undetected defects.
2. **Delayed Issue Detection**:
   - **Gap**: Issues are often detected late in the development cycle.
   - **Impact**: Increased cost and effort required to fix defects.
3. **Limited Automation**:
   - **Gap**: Limited or no automation in the validation process.
   - **Impact**: Slower development cycles and inefficient resource utilization.
4. **Inconsistent Compliance**:
   - **Gap**: Difficulty in maintaining consistent compliance with industry standards and regulations.
   - **Impact**: Potential legal and regulatory risks.
5. **Resource Intensive**:
   - **Gap**: Manual validation is resource-intensive, requiring significant time and effort from the development team.
   - **Impact**: Reduced productivity and higher operational costs.

Improvement Scope to Bridge the Capability Gaps

1. **Implement Automated Testing**:
   - **Action**: Develop and integrate automated test scripts for various validation scenarios (unit tests, integration tests, performance tests).
   - **Tools**: Use tools like Selenium, JUnit, TestNG, and Jenkins for automation.
   - **Benefit**: Ensures consistent and reliable validation, reducing manual effort and errors.
2. **Early Issue Detection**:
   - **Action**: Integrate continuous validation into the CI/CD pipeline to detect and resolve issues early.
   - **Tools**: Use CI tools like Jenkins, GitHub Actions, or GitLab CI.
   - **Benefit**: Reduces the risk of major defects and lowers the cost of fixing issues.
3. **Enhanced Automation**:
   - **Action**: Leverage automation tools to streamline the validation process and provide faster feedback.

   - **Tools**: Use tools like Selenium, JUnit, and TestNG.
   - **Benefit**: Increases efficiency and accelerates development cycles.
4. **Consistent Compliance**:
   - **Action**: Define and implement automated checks for compliance with industry standards and regulations.
   - **Tools**: Use tools like OWASP ZAP for security testing and compliance checks.
   - **Benefit**: Ensures consistent compliance and reduces legal and regulatory risks.
5. **Efficient Resource Utilization**:
   - **Action**: Automate the validation process to reduce manual effort and allow the team to focus on more critical tasks.
   - **Tools**: Use automation frameworks and CI/CD tools.
   - **Benefit**: Increases productivity and reduces operational costs.

Priority Order to Achieve Improvements

1. **Implement Automated Testing**:
   - **Priority**: High
   - **Reason**: Provides immediate benefits in terms of consistency, reliability, and reduced manual effort.
2. **Early Issue Detection**:
   - **Priority**: High
   - **Reason**: Critical for reducing the risk of major defects and lowering the cost of fixing issues.
3. **Enhanced Automation**:
   - **Priority**: Medium
   - **Reason**: Improves efficiency and accelerates development cycles, but requires initial setup and integration.
4. **Consistent Compliance**:
   - **Priority**: Medium
   - **Reason**: Ensures legal and regulatory compliance, reducing potential risks.
5. **Efficient Resource Utilization**:
   - **Priority**: Low
   - **Reason**: While important, it is a byproduct of implementing automated testing and enhanced automation.

By addressing these capability gaps in the specified priority order, you can effectively transition to continuous validation, leading to a more efficient, reliable, and compliant development process.

Would you like more details on any specific improvement action or tool?

Formatted: Font: Times New Roman, 14 pt

3.4 Roadmap of milestone and execution timeline

< Provide the details of the roadmap prepared for milestones defined, mechanism to track the execution of the milestone against the time and road ahead>

Answer:- **Roadmap for Transition to Continuous Validation**

Milestones Defined

1. **Initial Assessment and Planning**:
   - **Objective**: Conduct a thorough assessment of the current validation processes and identify key areas for improvement.
   - **Timeline**: 1 month
   - **Deliverables**: Detailed assessment report, implementation plan, resource allocation.
2. **Tool Selection and Integration**:
   - **Objective**: Evaluate and select appropriate automation and validation tools.
   - **Timeline**: 2 months
   - **Deliverables**: List of selected tools, integration plan, initial setup.
3. **Automated Test Development**:
   - **Objective**: Develop automated test scripts for various validation scenarios (unit tests, integration tests, performance tests).
   - **Timeline**: 3 months
   - **Deliverables**: Automated test scripts, test cases repository, initial test runs.
4. **CI/CD Pipeline Integration**:
   - **Objective**: Integrate continuous validation into the CI/CD pipeline.
   - **Timeline**: 2 months
   - **Deliverables**: Updated CI/CD pipeline, automated test execution, monitoring setup.
5. **Compliance and Security Checks**:
   - **Objective**: Implement automated checks for compliance with industry standards and security requirements.
   - **Timeline**: 2 months
   - **Deliverables**: Compliance check scripts, security validation reports, regular audits.
6. **Training and Support**:
   - **Objective**: Provide training to the development team on new validation processes and tools.
   - **Timeline**: 1 month
   - **Deliverables**: Training sessions, documentation, support system.
7. **Continuous Improvement**:
   - **Objective**: Regularly review and refine the continuous validation process.
   - **Timeline**: Ongoing
   - **Deliverables**: Updated validation processes, feedback incorporation, process optimization.

Mechanism to Track Execution of Milestones

1. **Project Management Tools**:
    - **Tools**: Use project management tools like Jira, Trello, or Asana to track progress.
    - **Features**: Task assignments, progress tracking, milestone tracking, and reporting.
2. **Regular Status Meetings**:
    - **Frequency**: Weekly or bi-weekly meetings to review progress and address any issues.
    - **Participants**: Project stakeholders, development team, QA team, and management.
3. **Progress Reports**:
    - **Frequency**: Monthly progress reports to provide updates on milestone achievements and any deviations from the plan.
    - **Content**: Status of each milestone, completed tasks, upcoming tasks, and any risks or issues.
4. **Dashboards and Metrics**:
    - **Tools**: Use dashboards in project management tools to visualize progress.
    - **Metrics**: Track key performance indicators (KPIs) such as test coverage, defect rates, and deployment frequency.
5. **Feedback Loops**:
    - **Mechanism**: Regular feedback sessions with the development and QA teams to gather insights and make necessary adjustments.
    - **Outcome**: Continuous improvement of the validation process based on feedback.

Road Ahead

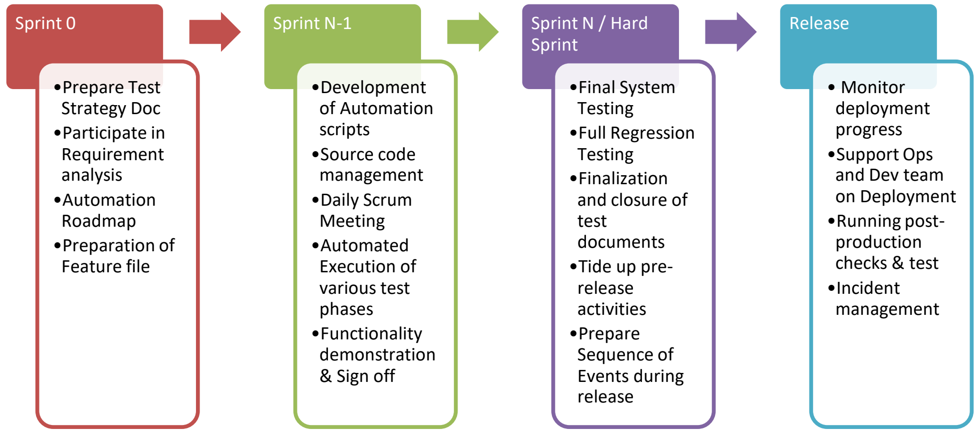
1. **Scalability**:
    - **Objective**: Ensure the validation process can scale with the application's growth.
    - **Actions**: Optimize automation scripts, enhance infrastructure, and continuously monitor performance.
2. **Advanced Automation**:
    - **Objective**: Implement advanced automation techniques such as AI-driven testing and self-healing scripts.
    - **Actions**: Research and integrate advanced automation tools, conduct pilot projects, and scale successful implementations.
3. **Enhanced Security**:
    - **Objective**: Strengthen security validation to protect against evolving threats.
    - **Actions**: Regularly update security checks, conduct penetration testing, and stay informed about new security practices.
4. **User Experience Focus**:
    - **Objective**: Improve the user experience by incorporating usability testing into the validation process.
    - **Actions**: Develop and integrate usability test cases, gather user feedback, and iterate on improvements.

By following this roadmap and tracking mechanisms, you can ensure a successful transition to continuous validation, leading to improved code quality, faster time-to-market, and enhanced customer satisfaction.
Would you like more details on any specific milestone or tracking mechanism?

Formatted: Font: Times New Roman, 14 pt

~~10.~~4. An illustrative list of activities of a continuous validation engineer

The following summarizes the typical activities to be done by DevOps team member during different sprints of the continuous delivery model project.


Sprint 0
•Prepare Test Strategy Doc
•Participate in Requirement analysis
•Automation Roadmap
•Preparation of Feature file
Sprint N-1
•Development of Automation scripts
•Source code management
•Daily Scrum Meeting
•Automated Execution of various test phases
•Functionality demonstration & Sign off
Sprint N / Hard Sprint
•Final System Testing
•Full Regression Testing
•Finalization and closure of test documents
•Tide up pre-release activities
•Prepare Sequence of Events during release
Release
• Monitor deployment progress
•Support Ops and Dev team on Deployment
•Running post-production checks & test
•Incident management

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

~~11.~~5. Continuous Validation – Building Pipeline

5.1 In Scope

< ***Provide the details of which phases of the testing (based on the architecture of the application) and the reason to be included in the continuous validation*** **pipeline** >

Formatted: Font: Times New Roman, 14 pt, Bold
Formatted: Font: Times New Roman, 14 pt

Answer:- **Phases of Testing to Include in the Continuous Validation Pipeline**

1. Unit Testing

- **Purpose**: To test individual components or units of the application in isolation.
- **Reason**: Unit tests are the fastest and least expensive to run. They ensure that each unit functions correctly, catching issues early in the development process[1].

2. Integration Testing

- **Purpose**: To test the interaction between integrated units or components.
- **Reason**: Ensures that combined components work together as expected, identifying issues that may arise from component interactions[1].

3. System Testing

- **Purpose**: To test the complete and integrated application as a whole.
- **Reason**: Validates the end-to-end functionality of the application, ensuring that all components work together seamlessly[2].

4. Regression Testing

- **Purpose**: To verify that recent changes have not adversely affected existing functionality.
- **Reason**: Ensures stability and reliability after updates, preventing new changes from breaking previously working features[2].

5. Performance Testing

- **Purpose**: To assess the application's performance under various conditions.
- **Reason**: Identifies bottlenecks and optimizes performance, ensuring the application can handle expected load and stress[2].

6. Security Testing

- **Purpose**: To identify vulnerabilities and ensure the application is secure.
- **Reason**: Protects against threats and ensures data integrity, maintaining compliance with security standards[3].

7. User Acceptance Testing (UAT)

- **Purpose**: To validate the application against user requirements and ensure it meets business needs.
- **Reason**: Ensures the application is acceptable to end-users and meets their expectations[3].

Reasons to Include These Phases in the Continuous Validation Pipeline

1. **Early Defect Detection**:
    - Continuous validation allows for early detection and resolution of issues, reducing the risk of major defects and lowering the cost of fixing issues[3].
2. **Improved Code Quality**:
    - By continuously validating code, the overall quality of the application is improved, leading to more stable and reliable releases[3].
3. **Faster Feedback Loops**:
    - Automated testing provides rapid feedback to developers, enabling quicker identification and resolution of issues[3].
4. **Enhanced Compliance**:
    - Continuous validation helps maintain compliance with industry standards and regulations by regularly checking code against these requirements[3].
5. **Increased Efficiency**:
    - Automating the validation process reduces the need for manual checks, allowing developers to focus on more value-added tasks[3].
6. **Better Alignment with Business Goals**:
    - Continuous validation ensures that the development efforts are aligned with business objectives by regularly reviewing and validating the strategy[3].

By including these testing phases in the continuous validation pipeline, you can ensure a comprehensive and efficient validation process, leading to higher quality and more reliable application releases.

Would you like more details on any specific testing phase or aspect of continuous validation?

[1]: 1 [2]: 2 [3]: 4

Formatted: Font: Times New Roman, 14 pt

5.1.1 Test Data Setup

< ***Provide the details around how test data set up will be automated. Also provide the details of tools being used, which activities to be carried (refresh database, running query etc)*** >

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answer:- **Automating Test Data Setup**

Automated Test Data Setup

1. **Data Generation**:
    - **Automated Tools**: Use tools like Tonic.ai, Mockaroo, or DataFactory to generate synthetic test data.
    - **Activities**: Create diverse datasets including valid, invalid, edge cases, and boundary values to ensure comprehensive test coverage[1].
2. **Data Management**:

  - **Automated Tools**: Use tools like Delphix, Informatica TDM, or CA Test Data Manager for managing test data.
  - **Activities**: Organize and store test data in a structured manner, ensuring easy access and version control[1].
3. **Data Masking and Anonymization**:
  - **Automated Tools**: Use tools like IBM InfoSphere Optim, DataVeil, or Oracle Data Masking.
  - **Activities**: Mask or anonymize sensitive data to protect privacy and comply with data protection regulations[1].
4. **Data Refresh and Cleanup**:
  - **Automated Tools**: Use scripts or tools like Jenkins, Ansible, or Chef to automate data refresh and cleanup.
  - **Activities**: Refresh databases, reset test data between test runs, and clean up test data after testing is complete[1].
5. **Data Validation**:
  - **Automated Tools**: Use tools like QuerySurge or custom scripts for data validation.
  - **Activities**: Perform automated checks to validate the correctness and integrity of test data[1].

Tools Used for Test Data Automation

1. **Tonic.ai**:
  - **Purpose**: Generate and manage synthetic test data.
  - **Features**: Data generation, masking, and variation[1].
2. **Mockaroo**:
  - **Purpose**: Generate realistic test data for various scenarios.
  - **Features**: Customizable data generation, support for multiple data types[1].
3. **Delphix**:
  - **Purpose**: Manage and provision test data.
  - **Features**: Data masking, data virtualization, and automated data refresh[1].
4. **Informatica TDM**:
  - **Purpose**: Test data management and provisioning.
  - **Features**: Data generation, masking, and subsetting[1].
5. **CA Test Data Manager**:
  - **Purpose**: Comprehensive test data management.
  - **Features**: Data generation, masking, and provisioning[1].

Activities to be Carried Out

1. **Refresh Database**:
  - **Tools**: Jenkins, Ansible, Chef.
  - **Activities**: Automate the process of refreshing the database to ensure it is up-to-date with the latest test data[1].
2. **Running Queries**:

- **Tools**: Custom scripts, SQL tools.
- **Activities**: Execute queries to validate data, generate reports, and ensure data integrity[1].

3. **Data Masking**:
   - **Tools**: IBM InfoSphere Optim, DataVeil.
   - **Activities**: Mask sensitive data to protect privacy and comply with regulations[1].
4. **Data Cleanup**:
   - **Tools**: Custom scripts, Jenkins.
   - **Activities**: Clean up test data after testing to maintain a consistent test environment[1].

By automating these activities and using the appropriate tools, you can ensure a robust and efficient test data setup process, leading to more reliable and comprehensive testing.
Would you like more details on any specific tool or activity? [1]: 1

Formatted: Font: Times New Roman, 14 pt

5.1.2 Smoke Testing

< Provide the details of which functionality is being tested during smoke testing. Also brief how specific functionality ensures the smoke test of the application. Provide the details of the tools being used for automating smoke test. >

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answer:- **Functionality Tested During Smoke Testing**

Key Functionalities Tested

1. **User Authentication**:
   - **Details**: Verifies that users can log in and log out successfully.
   - **Reason**: Ensures that the authentication mechanism is working correctly, which is critical for accessing the application.
2. **Navigation**:
   - **Details**: Checks that users can navigate through the main sections of the application.
   - **Reason**: Ensures that the application's navigation is functional and intuitive, allowing users to access different features.
3. **Data Input and Output**:
   - **Details**: Tests forms and data entry points to ensure data can be entered and retrieved correctly.
   - **Reason**: Validates that the application can handle user inputs and display outputs as expected.
4. **Core Functionalities**:
   - **Details**: Verifies that the primary features of the application (e.g., adding items to a cart, processing transactions) are working.
   - **Reason**: Ensures that the main functionalities are operational, which is essential for the application's purpose.
5. **Error Handling**:

   - **Details**: Checks that the application handles errors gracefully and provides appropriate error messages.
   - **Reason**: Ensures that users receive clear feedback when something goes wrong, improving the user experience.
6. **Database Connectivity**:
   - **Details**: Verifies that the application can connect to the database and perform basic CRUD (Create, Read, Update, Delete) operations.
   - **Reason**: Ensures that the application can interact with the database correctly, which is crucial for data integrity.

How Specific Functionality Ensures the Smoke Test of the Application

1. **User Authentication**:
   - **Ensures**: Validates that users can access the application securely, which is a fundamental requirement for any application.
2. **Navigation**:
   - **Ensures**: Confirms that users can move through the application without issues, which is essential for usability.
3. **Data Input and Output**:
   - **Ensures**: Verifies that the application can handle user data correctly, which is critical for functionality.
4. **Core Functionalities**:
   - **Ensures**: Confirms that the main features of the application are working, which is essential for the application’s purpose.
5. **Error Handling**:
   - **Ensures**: Validates that the application can handle errors gracefully, which is important for user experience.
6. **Database Connectivity**:
   - **Ensures**: Confirms that the application can interact with the database, which is crucial for data integrity and functionality.

Tools Used for Automating Smoke Tests

1. **Selenium**:
   - **Purpose**: Automates web application testing.
   - **Features**: Supports multiple browsers and platforms, integrates with CI/CD pipelines[1].
2. **Cypress**:
   - **Purpose**: End-to-end testing for web applications.
   - **Features**: Fast, reliable, and easy to set up, with real-time reloading[2].
3. **JUnit/TestNG**:
   - **Purpose**: Frameworks for writing and running repeatable automated tests in Java applications.
   - **Features**: Supports unit, integration, and smoke testing[2].

4. **Postman**:
    - **Purpose**: API testing and automation.
    - **Features**: Easy to create and run automated tests for APIs, integrates with CI/CD pipelines[2].

By using these tools and focusing on the key functionalities, you can ensure that your smoke tests effectively validate the stability and readiness of your application for further testing. Would you like more details on any specific tool or functionality? [1]: 1 [2]: 2

5.1.3 Functional Testing

< ***Brief about the functionalities to be tested during the continuous validation. Provide the details around tools being used to develop the automation scripts.*** >

Answer:- **Functionalities to be Tested During Continuous Validation**

Key Functionalities

1. **Core Business Logic**:
    - **Details**: Ensures that the main functionalities of the application, such as transaction processing, data calculations, and business rules, are working correctly.
    - **Reason**: Validates that the core operations of the application are reliable and accurate.
2. **User Interface (UI)**:
    - **Details**: Tests the responsiveness, usability, and accessibility of the application’s UI components.
    - **Reason**: Ensures a seamless and user-friendly experience across different devices and browsers.
3. **API Endpoints**:
    - **Details**: Validates the functionality, performance, and security of API endpoints.
    - **Reason**: Ensures that the application can communicate effectively with other services and systems.
4. **Database Operations**:
    - **Details**: Tests CRUD (Create, Read, Update, Delete) operations and data integrity.
    - **Reason**: Ensures that the application can interact with the database correctly and maintain data consistency.
5. **Security Features**:
    - **Details**: Validates authentication, authorization, and data encryption mechanisms.
    - **Reason**: Ensures that the application is secure and protects user data.
6. **Performance Metrics**:
    - **Details**: Assesses the application’s performance under various load conditions.
    - **Reason**: Identifies bottlenecks and optimizes performance to handle expected traffic.
7. **Error Handling**:
    - **Details**: Tests how the application handles errors and exceptions.
    - **Reason**: Ensures that the application provides meaningful error messages and recovers gracefully from failures.

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Tools Used for Developing Automation Scripts

1. **Selenium**:
    - **Purpose**: Automates web application testing.
    - **Features**: Supports multiple browsers and platforms, integrates with CI/CD pipelines[1].
2. **Cypress**:
    - **Purpose**: End-to-end testing for web applications.
    - **Features**: Fast, reliable, and easy to set up, with real-time reloading[1].
3. **JUnit/TestNG**:
    - **Purpose**: Frameworks for writing and running repeatable automated tests in Java applications.
    - **Features**: Supports unit, integration, and regression testing[1].
4. **Postman**:
    - **Purpose**: API testing and automation.
    - **Features**: Easy to create and run automated tests for APIs, integrates with CI/CD pipelines[1].
5. **Jenkins**:
    - **Purpose**: Continuous integration and continuous delivery (CI/CD) automation.
    - **Features**: Automates the build, test, and deployment processes, integrates with various testing tools[1].
6. **TestComplete**:
    - **Purpose**: Functional UI testing tool.
    - **Features**: Supports desktop, web, and mobile applications, integrates with CI/CD pipelines[1].

By leveraging these tools and focusing on the key functionalities, you can ensure that your continuous validation process is thorough, efficient, and reliable.

Would you like more details on any specific tool or functionality? [1]: 9

Formatted: Font: Times New Roman, 14 pt

5.1.4 Middleware testing through Automation and Virtualization

< ***Provide the details how SOA or Middleware testing will be carried through automation. Brief about which tool has been used for automation, virtualization and which services are virtualized to ensure middleware testing is fully covered.*** >

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answer:- **Automating SOA and Middleware Testing**

Automation of SOA and Middleware Testing

1. **Service-Oriented Architecture (SOA) Testing**:
    - **Objective**: Validate the interactions between services, ensuring they work together as expected.
    - **Approach**: Automate the testing of individual services and their integrations using specialized tools.

2. **Middleware Testing**:
    - **Objective**: Ensure that the middleware components (e.g., message brokers, application servers) function correctly and efficiently.
    - **Approach**: Automate the testing of middleware interactions, message flows, and data transformations.

Tools Used for Automation

1. **Parasoft SOAtest**:
    - **Purpose**: Comprehensive testing of SOA and APIs.
    - **Features**: Supports functional, security, and performance testing of web services and APIs[1].
2. **SoapUI**:
    - **Purpose**: Functional testing of web services.
    - **Features**: Supports SOAP and REST protocols, easy to create and run automated tests[1].
3. **IBM Rational Integration Tester**:
    - **Purpose**: Automated testing of complex integrations.
    - **Features**: Supports testing of middleware, APIs, and message queues[2].
4. **Apache JMeter**:
    - **Purpose**: Performance and load testing.
    - **Features**: Supports testing of web applications, APIs, and various protocols[2].
5. **Postman**:
    - **Purpose**: API testing and automation.
    - **Features**: Easy to create and run automated tests for APIs, integrates with CI/CD pipelines[2].

Virtualization for Middleware Testing

1. **Service Virtualization**:
    - **Objective**: Simulate the behavior of dependent components or services to enable testing in isolation.
    - **Tools**:
        - **CA Service Virtualization**: Simulates services, databases, and mainframes to enable testing without dependencies[3].
        - **Parasoft Virtualize**: Creates virtual services to simulate the behavior of dependent components[4].
2. **Virtualized Services**:
    - **API Endpoints**: Simulate API responses to test how the application handles different scenarios.
    - **Database Services**: Virtualize database interactions to test data retrieval and storage without affecting the actual database.
    - **Message Queues**: Simulate message queues to test message flows and processing.

- **Third-Party Services**: Virtualize external services to test integrations without relying on the actual third-party systems.

By using these tools and virtualization techniques, you can ensure comprehensive and efficient testing of SOA and middleware components, leading to more reliable and robust applications.

Would you like more details on any specific tool or aspect of SOA and middleware testing? [1]: 5 [2]: 6 [4]: 11

5.1.5 Performance Testing

< ***Provide the details of which performance testing aspects – Stress, Load and Endurance. Provide the details around concurrent users, peak load and tool being used for the continuous validation. Brief about how the result will impact the functionality of an application*** >

Answer:- **Performance Testing Aspects**

1. Stress Testing

- **Purpose**: Evaluates the application's behavior under extreme load conditions, beyond its normal operational capacity.
- **Objective**: Identify the breaking point and how the application recovers from failure.
- **Example**: Simulating a sudden spike in user traffic to see how the application handles the load.

2. Load Testing

- **Purpose**: Measures the application's performance under expected load conditions.
- **Objective**: Ensure the application can handle the anticipated number of concurrent users and transactions.
- **Example**: Simulating the average number of users accessing the application during peak hours.

3. Endurance Testing

- **Purpose**: Assesses the application's performance over an extended period.
- **Objective**: Detect issues like memory leaks or performance degradation over time.
- **Example**: Running the application continuously for several hours or days to monitor its stability and performance[12].

Concurrent Users and Peak Load

- **Concurrent Users**: The number of users accessing the application simultaneously. This is crucial for understanding how the application performs under real-world usage scenarios.
- **Peak Load**: The maximum number of concurrent users or transactions the application can handle without performance degradation.

Tools Used for Continuous Validation

1. **Apache JMeter**:
   - **Purpose**: Performance and load testing.

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

  - **Features**: Simulates multiple users, measures response times, and identifies performance bottlenecks[1].

2. **LoadRunner**:
   - **Purpose**: Load testing and performance monitoring.
   - **Features**: Supports a wide range of protocols, provides detailed performance metrics[1].
3. **Gatling**:
   - **Purpose**: Load and stress testing.
   - **Features**: High performance, easy-to-use DSL for test scripting[1].

Impact of Results on Application Functionality

1. **Identifying Bottlenecks**:
   - **Impact**: Helps pinpoint areas where the application struggles under load, allowing for targeted optimizations.
2. **Ensuring Scalability**:
   - **Impact**: Validates that the application can scale to handle increased user load, ensuring it can grow with user demand.
3. **Improving Stability**:
   - **Impact**: Detects issues like memory leaks or performance degradation over time, leading to a more stable application.
4. **Enhancing User Experience**:
   - **Impact**: Ensures that the application remains responsive and reliable under various load conditions, improving overall user satisfaction.

By incorporating these performance testing aspects and using the appropriate tools, you can ensure that your application is robust, scalable, and capable of delivering a high-quality user experience.

Would you like more details on any specific aspect of performance testing? [1]: 1 [2]: 7

Formatted: Font: Times New Roman, 14 pt

5.2 Tool Selection Strategy

< ***Provide the details of technology fitment with the tool, commercial aspect (open source vs licensed), the tools being considered, comparison factors and final conclusion of the tools being selected.*** >

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answer::- **Technology Fitment and Tool Selection**

Technology Fitment with the Tool

1. **Compatibility**:
   - **Details**: Ensure the tool supports the technologies used in your application (e.g., programming languages, frameworks, databases).
   - **Example**: Selenium is suitable for web applications, while Appium is better for mobile applications.

2. **Integration**:
    - **Details**: Check if the tool integrates well with your existing CI/CD pipeline and other tools.
    - **Example**: Jenkins integrates seamlessly with various testing tools like Selenium, JUnit, and Postman.
3. **Scalability**:
    - **Details**: Ensure the tool can handle the scale of your application and testing needs.
    - **Example**: JMeter can simulate thousands of users for load testing.

Commercial Aspect: Open Source vs. Licensed

1. **Open Source Tools**:
    - **Pros**:
        - Cost-effective (free to use).
        - Community support and frequent updates.
        - Flexibility to customize.
    - **Cons**:
        - May require more technical expertise.
        - Limited official support.
        - Potential security risks if not managed properly.
2. **Licensed Tools**:
    - **Pros**:
        - Professional support and maintenance.
        - Comprehensive features and integrations.
        - Compliance with industry standards.
    - **Cons**:
        - Higher cost (license fees).
        - Vendor lock-in.
        - May have less flexibility for customization.

Tools Being Considered

1. **Selenium (Open Source)**:
    - **Purpose**: Web application testing.
    - **Features**: Supports multiple browsers and platforms, integrates with CI/CD tools[1].
2. **LoadRunner (Licensed)**:
    - **Purpose**: Performance and load testing.
    - **Features**: Supports a wide range of protocols, detailed performance metrics[1].
3. **Postman (Open Source)**:
    - **Purpose**: API testing and automation.
    - **Features**: Easy to create and run automated tests, integrates with CI/CD pipelines[1].
4. **TestComplete (Licensed)**:
    - **Purpose**: Functional UI testing.

  - **Features**: Supports desktop, web, and mobile applications, integrates with CI/CD pipelines[1].

Comparison Factors

1. **Cost**:
   - **Open Source**: Free to use, but may incur costs for setup and maintenance.
   - **Licensed**: Upfront and ongoing license fees, but includes support and updates.
2. **Support**:
   - **Open Source**: Community support, which can be variable.
   - **Licensed**: Professional support with SLAs.
3. **Features**:
   - **Open Source**: Basic features with potential for customization.
   - **Licensed**: Comprehensive features out-of-the-box.
4. **Ease of Use**:
   - **Open Source**: May require more technical expertise.
   - **Licensed**: Generally easier to use with better documentation and support.
5. **Security**:
   - **Open Source**: Requires careful management to ensure security.
   - **Licensed**: Often includes built-in security features and compliance.

Final Conclusion

Based on the comparison, the choice between open source and licensed tools depends on your specific needs and constraints:

- **For Cost-Effectiveness and Flexibility**: Open source tools like Selenium and Postman are ideal if you have the technical expertise to manage and customize them.
- **For Comprehensive Features and Support**: Licensed tools like LoadRunner and TestComplete are better if you need professional support, compliance, and a wide range of features out-of-the-box.

Ultimately, a hybrid approach, leveraging both open source and licensed tools, can provide the best of both worlds, ensuring cost-effectiveness, flexibility, and comprehensive support.

Would you like more details on any specific tool or aspect of this comparison? [1]: 1

Formatted: Font: Times New Roman, 14 pt

5.3 Test Automation Roadmap

< Provide the detailed plan of automation script to be delivered over various testing phases against the sprints. >

Answer:- **Detailed Plan for Automation Script Delivery Across Testing Phases and Sprints**

Sprint 1: Initial Setup and Unit Testing

- **Activities**:

  - Set up the automation framework (e.g., Selenium, JUnit).
  - Develop initial unit test scripts for core functionalities.
  - Integrate unit tests into the CI/CD pipeline.
- **Deliverables**:
  - Automation framework setup.
  - Unit test scripts for core functionalities.
  - CI/CD integration for unit tests.

Sprint 2: Integration Testing

- **Activities**:
  - Develop integration test scripts to validate interactions between components.
  - Automate integration tests and integrate them into the CI/CD pipeline.
  - Ensure data setup and teardown processes are automated.
- **Deliverables**:
  - Integration test scripts.
  - Automated data setup and teardown scripts.
  - CI/CD integration for integration tests.

Sprint 3: System Testing

- **Activities**:
  - Develop system test scripts to validate end-to-end functionality.
  - Automate system tests and integrate them into the CI/CD pipeline.
  - Perform initial system testing and refine scripts based on feedback.
- **Deliverables**:
  - System test scripts.
  - CI/CD integration for system tests.
  - Refined system test scripts.

Sprint 4: Regression Testing

- **Activities**:
  - Identify critical test cases for regression testing.
  - Develop and automate regression test scripts.
  - Integrate regression tests into the CI/CD pipeline.
- **Deliverables**:
  - Regression test scripts.
  - CI/CD integration for regression tests.
  - Comprehensive regression test suite.

Sprint 5: Performance Testing

- **Activities**:
  - Develop performance test scripts to assess load, stress, and endurance.
  - Automate performance tests using tools like JMeter or LoadRunner.
  - Integrate performance tests into the CI/CD pipeline.
- **Deliverables**:

  - Performance test scripts.
  - CI/CD integration for performance tests.
  - Initial performance test results and analysis.

Sprint 6: Security Testing

- **Activities**:
  - Develop security test scripts to identify vulnerabilities.
  - Automate security tests using tools like OWASP ZAP or Burp Suite.
  - Integrate security tests into the CI/CD pipeline.
- **Deliverables**:
  - Security test scripts.
  - CI/CD integration for security tests.
  - Initial security test results and analysis.

Sprint 7: User Acceptance Testing (UAT)

- **Activities**:
  - Develop UAT scripts based on user requirements.
  - Automate UAT scripts and integrate them into the CI/CD pipeline.
  - Perform UAT and gather feedback from stakeholders.
- **Deliverables**:
  - UAT scripts.
  - CI/CD integration for UAT.
  - UAT feedback and refinement.

Tools Being Used for Automation

1. **Selenium**:
   - **Purpose**: Web application testing.
   - **Features**: Supports multiple browsers and platforms, integrates with CI/CD tools[1].
2. **JUnit/TestNG**:
   - **Purpose**: Unit and integration testing.
   - **Features**: Supports repeatable automated tests, integrates with CI/CD pipelines[1].
3. **Postman**:
   - **Purpose**: API testing and automation.
   - **Features**: Easy to create and run automated tests for APIs, integrates with CI/CD pipelines[1].
4. **JMeter**:
   - **Purpose**: Performance testing.
   - **Features**: Simulates multiple users, measures response times, identifies performance bottlenecks[1].
5. **OWASP ZAP**:
   - **Purpose**: Security testing.
   - **Features**: Identifies vulnerabilities, integrates with CI/CD pipelines[1].

Tracking and Execution

1. **Project Management Tools**:
    - **Tools**: Jira, Trello, Asana.
    - **Features**: Task assignments, progress tracking, milestone tracking, and reporting.
2. **CI/CD Integration**:
    - **Tools**: Jenkins, GitHub Actions, GitLab CI.
    - **Features**: Automates the build, test, and deployment processes, integrates with various testing tools.
3. **Regular Status Meetings**:
    - **Frequency**: Weekly or bi-weekly meetings to review progress and address any issues.
    - **Participants**: Project stakeholders, development team, QA team, and management.
4. **Progress Reports**:
    - **Frequency**: Monthly progress reports to provide updates on milestone achievements and any deviations from the plan.
    - **Content**: Status of each milestone, completed tasks, upcoming tasks, and any risks or issues.

By following this detailed plan and using the appropriate tools, you can ensure a comprehensive and efficient automation script delivery process across various testing phases and sprints.

Would you like more details on any specific phase or tool? [1]: 1

Formatted: Font: Times New Roman, 14 pt

< Update the following table summarizes the automation roadmap for the project >

Sprint	Objective of Test Automation	Target Date	Overall %	Comments

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Here's an updated table summarizing the automation roadmap for your project:

Sprint	***Objective of Test Automation***	***Target Date***	***Overall %***	***Comments***
Sprint 1	Initial Setup and Unit Testing	2024-08-15	10%	Set up the automation framework and develop initial unit test scripts.
Sprint 2	Integration Testing	2024-08-30	20%	Develop and automate integration test scripts, automate data setup and teardown.
Sprint 3	System Testing	2024-09-15	30%	Develop and automate system test scripts, perform initial system testing.
Sprint 4	Regression Testing	2024-09-30	40%	Identify critical test cases, develop and automate regression test scripts.
Sprint 5	Performance Testing	2024-10-15	50%	Develop and automate

Formatted: Font: Italic

Sprint	***Objective of Test Automation***	***Target Date***	***Overall %***	***Comments***
				performance test scripts, integrate into CI/CD pipeline.
Sprint 6	Security Testing	2024-10-30	60%	Develop and automate security test scripts, perform initial security testing.
Sprint 7	User Acceptance Testing (UAT)	2024-11-15	70%	Develop and automate UAT scripts, gather feedback from stakeholders.
Sprint 8	Continuous Improvement	2024-11-30	80%	Refine and optimize test scripts based on feedback and performance data.
Sprint 9	Full Integration	2024-12-15	90%	Ensure all tests are

Formatted: Font: Italic

Sprint	***Objective of Test Automation***	***Target Date***	***Overall %***	***Comments***
	and Final Testing			integrated and perform final testing.
Sprint 10	Final Review and Documentation	2024-12-31	100%	Review all test scripts, finalize documentation, and prepare for deployment.

This roadmap ensures a structured and phased approach to developing and integrating automation scripts across various testing phases, aligned with your project sprints. Would you like more details on any specific sprint or phase?

5.4 Test Environment Strategy

< ***Provide the details around the environments available for testing or pre-production. Brief about how code movement will happen from one environment to another post successful completion of the test phase*** >

Answer:- Environments Available for Testing and Pre-Production

1. Development Environment

- **Purpose: Used by developers to write and initially test their code.**
- **Activities: Unit testing, initial integration testing.**
- **Tools: IDEs, local databases, version control systems.**

2. Test Environment

- **Purpose: Used by QA teams to perform various types of testing.**
- **Activities: Functional testing, integration testing, regression testing.**

Formatted: Font: Italic

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

- **Tools: Test automation tools (e.g., Selenium, JUnit), test management tools (e.g., Jira, TestRail).**

3. Staging Environment

- **Purpose: A replica of the production environment used for final testing before deployment.**
- **Activities: User acceptance testing (UAT), performance testing, security testing.**
- **Tools: Monitoring tools (e.g., Prometheus, Grafana), performance testing tools (e.g., JMeter, LoadRunner).**

4. Pre-Production Environment

- **Purpose: Used for final validation and testing before code is moved to production.**
- **Activities: End-to-end testing, final regression testing, compliance checks.**
- **Tools: Deployment tools (e.g., Jenkins, Ansible), security testing tools (e.g., OWASP ZAP).**

Code Movement from One Environment to Another

1. **Development to Test Environment:**
    - **Process: Once developers complete their code and initial unit tests, the code is committed to the version control system (e.g., Git).**
    - **Automation: CI tools like Jenkins or GitHub Actions automatically build the code and run initial tests.**
    - **Deployment: If the build and tests pass, the code is deployed to the test environment for further testing[1].**
2. **Test to Staging Environment:**
    - **Process: After successful testing in the test environment, the code is tagged and merged into the staging branch.**
    - **Automation: CI/CD tools trigger automated builds and deployments to the staging environment.**

   - **Deployment: The code undergoes UAT, performance, and security testing in the staging environment[1].**

3. **Staging to Pre-Production Environment:**
   - **Process: Once the code passes all tests in the staging environment, it is prepared for pre-production deployment.**
   - **Automation: Deployment scripts and tools ensure the code is deployed to the pre-production environment.**
   - **Deployment: Final validation and compliance checks are performed in the pre-production environment[1].**
4. **Pre-Production to Production Environment:**
   - **Process: After successful validation in the pre-production environment, the code is approved for production deployment.**
   - **Automation: CI/CD pipelines automate the deployment process to the production environment.**
   - **Deployment: The code is deployed to the production environment, and monitoring tools ensure it runs smoothly[1].**

By following this structured approach and using appropriate tools, you can ensure a smooth and efficient code movement process across different environments, leading to reliable and high-quality software releases.

Would you like more details on any specific environment or deployment process? [1]: 5

5.4.1 Test Environment / Pre-production Environment

< Brief about the test environment details. The user access, rights and limitation of the environment. Provide which checks will be carried post deployment of the code to ensure health of the environment is okay to move testing forward >

Answer:**- Test Environment Details**

1. Development Environment

- **Purpose**: Used by developers for writing and initially testing their code.

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

- **User Access**: Developers have full access to modify and test their code.
- **Rights and Limitations**: Full administrative rights, but changes here do not affect other environments.

2. Test Environment

- **Purpose**: Used by QA teams to perform various types of testing.
- **User Access**: QA engineers and testers have access to execute tests and report issues.
- **Rights and Limitations**: Limited to testing activities; no code modifications allowed. Access to test data and test scripts.

3. Staging Environment

- **Purpose**: A replica of the production environment for final testing before deployment.
- **User Access**: QA engineers, testers, and sometimes product owners for UAT.
- **Rights and Limitations**: Limited to testing and validation; no code changes. Access to staging data and configurations.

4. Pre-Production Environment

- **Purpose**: Used for final validation and testing before code is moved to production.
- **User Access**: QA engineers, testers, and release managers.
- **Rights and Limitations**: Limited to final testing and validation; no code changes. Access to pre-production data and configurations.

User Access, Rights, and Limitations

1. **User Access**:
   - **Developers**: Full access in the development environment, limited access in test and staging environments.
   - **QA Engineers**: Access to test and staging environments for executing tests and reporting issues.
   - **Product Owners**: Access to staging and pre-production environments for UAT.
   - **Release Managers**: Access to all environments for overseeing deployments.
2. **Rights and Limitations**:
   - **Development Environment**: Full rights for code changes and testing.
   - **Test Environment**: Rights to execute tests and access test data; no code modifications.
   - **Staging Environment**: Rights to perform UAT and validate changes; no code modifications.
   - **Pre-Production Environment**: Rights to perform final validation and compliance checks; no code modifications.

Post-Deployment Checks

1. **Smoke Testing**:
   - **Purpose**: Verify that the major functionalities of the application are working correctly.
   - **Activities**: Run automated smoke tests to ensure basic functionality is intact.
2. **Health Checks**:

  - **Purpose**: Ensure the environment is stable and all services are running.
  - **Activities**: Monitor system logs, check service statuses, and verify database connectivity.
3. **Performance Monitoring**:
  - **Purpose**: Assess the application's performance under the current load.
  - **Activities**: Use monitoring tools like Prometheus and Grafana to track performance metrics.
4. **Security Scans**:
  - **Purpose**: Identify any security vulnerabilities introduced by the new code.
  - **Activities**: Run automated security scans using tools like OWASP ZAP or Burp Suite.
5. **Regression Testing**:
  - **Purpose**: Ensure that new changes have not adversely affected existing functionality.
  - **Activities**: Execute automated regression test suites to validate the stability of the application.

By following these steps and using appropriate tools, you can ensure that the test environment is healthy and ready for further testing, leading to reliable and high-quality software releases.

Would you like more details on any specific environment or post-deployment check? [1]: 1 [2]: 2 [3]: 5

5.4.2 Production Environment

< ***Brief about the test environment details. The user access, rights and limitation of the environment. Provide which checks will be carried post deployment of the code to ensure health of the environment is okay***>

Answer:- Sure! Here's a brief overview of the test environment details, user access, rights and limitations, and the checks to be carried out post-deployment:

Test Environment Details

A test environment is a dedicated setup where software testing activities are conducted. It replicates the production environment to ensure accurate and reliable testing. This environment includes hardware, software, network configurations, databases, and test data[1].

User Access, Rights, and Limitations

1. **User Access**:
  - **Developers**: Typically have access to deploy code, run tests, and debug issues.
  - **Testers**: Have access to execute test cases, report bugs, and verify fixes.
  - **Administrators**: Have full access to manage the environment, including setting up and maintaining configurations.
2. **Rights and Limitations**:
  - **Read/Write Access**: Developers and testers usually have read/write access to specific directories and databases relevant to their tasks.

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

  - **Restricted Access**: Sensitive areas, such as security configurations and critical system files, are restricted to administrators only.
  - **Role-Based Access Control (RBAC)**: Ensures that users have the minimum necessary access to perform their roles, reducing the risk of unauthorized changes[2].

Post-Deployment Checks

After deploying code to the test environment, several checks are performed to ensure the environment's health:

1. **Smoke Testing**: A quick set of tests to verify that the major functionalities are working correctly.
2. **Regression Testing**: Ensures that new changes haven't adversely affected existing functionalities.
3. **Performance Testing**: Checks the system's performance under various conditions to ensure it meets the required standards.
4. **Security Testing**: Identifies vulnerabilities and ensures that security measures are effective.
5. **Environment Monitoring**: Continuous monitoring of system resources (CPU, memory, disk usage) to detect any anomalies.
6. **Log Analysis**: Reviewing logs for errors or warnings that might indicate issues with the deployment.

These steps help maintain a stable and reliable test environment, ensuring that any issues are identified and addressed promptly.

Is there anything specific you'd like to know more about?

Formatted: Font: Times New Roman, 14 pt

5.5 Test Execution and Test Automation Strategy

< Provide the details of which critical functionalities will be executed and how various aspects will be taken care i.e. Defect management, updating test case results in test management tools >

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answer:**- Critical Functionalities to be Executed**

1. **User Authentication and Authorization**:
   - Verify login, logout, and session management.
   - Ensure role-based access control (RBAC) is functioning correctly.
2. **Data Integrity and Transactions**:
   - Test CRUD (Create, Read, Update, Delete) operations.
   - Validate data consistency across different modules.
3. **API Endpoints**:
   - Check the functionality of all exposed APIs.
   - Ensure proper error handling and response times.
4. **User Interface (UI) Functionality**:
   - Validate navigation, form submissions, and user interactions.

- Ensure responsiveness across different devices and browsers.

5. **Performance Metrics**:
    - Measure load times, throughput, and resource utilization.
    - Conduct stress and load testing to identify bottlenecks.

Defect Management

1. **Defect Logging**:
    - Use tools like JIRA, Bugzilla, or Azure DevOps to log defects.
    - Include detailed steps to reproduce, screenshots, and severity levels.
2. **Defect Triage**:
    - Prioritize defects based on impact and urgency.
    - Assign defects to the appropriate developers for resolution.
3. **Defect Tracking**:
    - Monitor the status of defects from identification to closure.
    - Conduct regular defect review meetings to ensure timely resolution.

Updating Test Case Results in Test Management Tools

1. **Test Execution**:
    - Execute test cases using tools like TestRail, HP ALM, or Zephyr.
    - Record the results (Pass/Fail) for each test case.
2. **Result Documentation**:
    - Attach evidence (screenshots, logs) to the test cases in the tool.
    - Provide comments or notes for any deviations or observations.
3. **Reporting**:
    - Generate test execution reports to track progress and coverage.
    - Share reports with stakeholders to keep them informed about the testing status.

Ensuring Comprehensive Coverage

1. **Test Case Review**:
    - Regularly review and update test cases to cover new functionalities and edge cases.
    - Ensure test cases are aligned with the latest requirements and specifications.
2. **Continuous Integration (CI)**:
    - Integrate automated tests into the CI pipeline using tools like Jenkins, GitLab CI, or Azure Pipelines.
    - Ensure tests are executed with every code commit to catch issues early.
3. **Feedback Loop**:
    - Maintain open communication between developers, testers, and stakeholders.
    - Use feedback to improve test cases, processes, and overall quality.

These steps ensure that critical functionalities are thoroughly tested, defects are managed efficiently, and test results are accurately documented, leading to a robust and reliable software product.

Is there anything specific you’d like to dive deeper into?

Formatted: Font: Times New Roman, 14 pt

< ***Also brief about the test automation tool, status of the scripts will be updated and the critical functionalities to be prioritized during continuous validation*** >

Answer:- **Test Automation Tool**

One of the popular test automation tools is **Selenium**. It's widely used for automating web applications for testing purposes. Here are some key features:

- **Cross-Browser Testing**: Supports multiple browsers like Chrome, Firefox, Safari, and Edge.
- **Language Support**: Compatible with various programming languages such as Java, Python, C#, and Ruby.
- **Integration**: Easily integrates with CI/CD tools like Jenkins, GitLab CI, and Azure Pipelines.
- **Flexibility**: Allows for the creation of complex test scripts to handle various testing scenarios[1].

Updating the Status of Scripts

1. **Automated Execution**:
    - Scripts are executed automatically as part of the CI/CD pipeline.
    - Results are captured and logged in real-time.
2. **Reporting**:
    - Use tools like Allure or ExtentReports to generate detailed test reports.
    - Reports include pass/fail status, execution time, and error logs.
3. **Dashboard**:
    - Implement a dashboard (e.g., using Grafana or Kibana) to visualize test results and trends.
    - Provides a quick overview of the test execution status and helps in identifying patterns.

Critical Functionalities to Prioritize During Continuous Validation

1. **Core Business Logic**:
    - Ensure that the main functionalities, such as payment processing or user registration, are always working correctly.
2. **API Endpoints**:
    - Continuously validate the functionality and performance of critical APIs.
    - Check for response times, data integrity, and error handling.
3. **User Authentication**:
    - Regularly test login, logout, and session management features.
    - Ensure security measures like multi-factor authentication are functioning properly.
4. **Data Integrity**:
    - Validate that data is correctly stored, retrieved, and updated across different modules.
    - Perform consistency checks to ensure data accuracy.
5. **Performance Metrics**:

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

    - Monitor key performance indicators such as load times, throughput, and resource utilization.
    - Conduct regular performance tests to identify and address bottlenecks.
6. **Security**:
    - Continuously test for vulnerabilities and ensure that security patches are applied promptly.
    - Validate that access controls and data encryption are effective.

By focusing on these critical functionalities and maintaining an updated status of test scripts, you can ensure a robust and reliable software product throughout the development lifecycle. Would you like more details on any specific aspect? [1]: Test Automation Tools

Formatted: Font: Times New Roman, 14 pt

5.6 Defect Management Strategy

< ***Provide the details of how defects will be logged to defect management tool. Also brief how defect would be treated over a defect life cycle and communication mechanism to notify of in the case of defect status change*** >

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answers:- **Logging Defects in a Defect Management Tool**

1. **Defect Identification**:
    - When a defect is identified, it is logged into a defect management tool like **JIRA**, **Bugzilla**, or **Azure DevOps**.
    - The defect report includes details such as defect ID, summary, description, steps to reproduce, expected and actual results, severity, priority, and attachments (e.g., screenshots, logs).
2. **Defect Logging Process**:
    - **Create a New Defect**: Navigate to the defect management tool and select the option to create a new defect.
    - **Fill in Details**: Enter all relevant information, including the environment where the defect was found, the module or feature affected, and any additional notes.
    - **Assign the Defect**: Assign the defect to the appropriate developer or team for resolution.
    - **Save and Submit**: Save the defect report, which will then be added to the defect tracking system.

Defect Life Cycle

1. **New**:
    - The defect is logged and awaits review.
    - Initial triage is performed to validate the defect and determine its severity and priority.
2. **Assigned**:
    - The defect is assigned to a developer or team for investigation and resolution.

- The assignee reviews the defect details and starts working on a fix.

3. **In Progress**:
    - The developer works on fixing the defect.
    - The status is updated to reflect that the defect is being actively addressed.
4. **Fixed**:
    - Once the defect is resolved, the status is updated to “Fixed.”
    - The fix is deployed to a test environment for verification.
5. **Retest**:
    - Testers verify the fix by retesting the defect.
    - If the defect is resolved, it moves to the next stage; if not, it is reassigned to the developer.
6. **Closed**:
    - If the defect is successfully fixed and verified, it is marked as “Closed.”
    - The defect is considered resolved and no further action is required.
7. **Reopened**:
    - If the defect reoccurs or the fix is not satisfactory, it is reopened and reassigned for further investigation.

Communication Mechanism for Defect Status Changes

1. **Automated Notifications**:
    - The defect management tool sends automated email notifications to relevant stakeholders whenever there is a status change.
    - Notifications include details about the defect, the new status, and any comments or updates.
2. **Dashboard and Reports**:
    - Stakeholders can view real-time status updates through dashboards and reports generated by the defect management tool.
    - These dashboards provide a comprehensive view of all defects, their statuses, and progress.
3. **Regular Meetings**:
    - Conduct regular defect triage and review meetings to discuss the status of critical defects.
    - Ensure open communication between developers, testers, and project managers to address any issues promptly.
4. **Comments and Mentions**:
    - Use the commenting feature in the defect management tool to add notes, updates, and mentions.
    - Tag relevant team members to ensure they are notified of important changes or actions required.

By following these processes, you can ensure that defects are logged, tracked, and resolved efficiently, with clear communication and visibility throughout the defect life cycle.

Is there anything specific you'd like to explore further?

5.7 Setting up Automated Gating criteria

< ***Provide the details of which all critical functionalities must be successfully tested in order to promote the code to the next level or next phase of the continuous validation pipeline. Brief which tool being used and how automated gating criteria would be set up*** >

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt, Bold

Formatted: Font: Times New Roman, 14 pt

Answers:- **Critical Functionalities for Code Promotion**

To ensure a smooth promotion of code to the next phase in the continuous validation pipeline, the following critical functionalities must be successfully tested:

1. **User Authentication and Authorization**:
   - Verify login, logout, and session management.
   - Ensure role-based access control (RBAC) is functioning correctly.
2. **Core Business Logic**:
   - Validate key business processes and workflows.
   - Ensure data integrity and consistency across modules.
3. **API Endpoints**:
   - Test the functionality, performance, and security of all exposed APIs.
   - Ensure proper error handling and response times.
4. **User Interface (UI) Functionality**:
   - Validate navigation, form submissions, and user interactions.
   - Ensure responsiveness across different devices and browsers.
5. **Performance Metrics**:
   - Measure load times, throughput, and resource utilization.
   - Conduct stress and load testing to identify bottlenecks.
6. **Security**:
   - Perform security testing to identify vulnerabilities.
   - Ensure compliance with security standards and best practices.

Tool Being Used

A popular tool for test automation in CI/CD pipelines is **Selenium**. It is widely used for automating web applications and supports multiple browsers and programming languages.

Automated Gating Criteria Setup

Automated gating criteria ensure that only code meeting predefined quality standards is promoted to the next phase. Here's how to set it up:

1. **Define Gating Criteria**:
   - **Unit Tests**: All unit tests must pass.
   - **Integration Tests**: Critical integration tests must pass.
   - **End-to-End Tests**: Key end-to-end tests must pass.

  - **Performance Benchmarks**: Performance tests must meet specified thresholds.
  - **Security Checks**: No critical security vulnerabilities should be present.
2. **Implement in CI/CD Pipeline**:
   - Use a CI/CD tool like **Jenkins**, **GitLab CI**, or **Azure Pipelines** to automate the process.
   - Configure the pipeline to run the defined tests and checks at each stage.
3. **Automated Approval**:
   - Set up automated gates that check the results of the tests.
   - If all criteria are met, the code is automatically promoted to the next phase.
   - If any criteria fail, the pipeline stops, and notifications are sent to the relevant stakeholders.
4. **Continuous Monitoring**:
   - Use monitoring tools like **Grafana** or **Kibana** to visualize test results and trends.
   - Continuously monitor the health of the pipeline and make adjustments as needed.

By ensuring these critical functionalities are thoroughly tested and setting up automated gating criteria, you can maintain a high level of quality and reliability in your software development process[1][2][3].

Would you like more details on any specific aspect? [1]: Fast track code promotion in your CI/CD pipeline [2]: From Code to Production: A Comprehensive Practical Guide to CI/CD Pipelines [3]: Continuous Validation: What Is It And Why Is It Important?

Formatted: Font: Times New Roman, 14 pt

5.8 Reporting & Test Metrics

Following key metrics will be captured as part of the metrics dashboard.

Metric	**Definition**	**Frequency**
Defect Acceptance Ratio (DAR)	Defect Acceptance is % ratio of number of defects accepted to the total number of defects raised.	
Defect Removal Efficiency (DRE)	DRE is a ratio of defects raised against the sum of post-production defects and total number of defects raised.	
% Deferred Defects	Deferred defects is a ratio of sum of Open Defects and deferred defects against total number of defects	
Reopen defects	Reopen defects is a count of defects reopened after retesting	

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Test case Execution (TCE)	% ratio of test cases executed vs. test cases planned	
Test Automation Coverage	Total Test Case Actually Automated (For Application)/Total Test Cases feasible to automate	
Execution cycle time in continuous validation	Time taken to execute all the scripts in a particular test phase of continuous validation	

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Formatted: Font: Times New Roman, 14 pt

Reporting & Test Metrics

Here are the key metrics that will be captured as part of the metrics dashboard:

Formatted: Font: Italic

Formatted Table

Metric	***Definition***	***Frequency***
Defect Acceptance Ratio (DAR)	Defect Acceptance is the percentage ratio of the number of defects accepted to the total number of defects raised.	Continuous
Defect Removal Efficiency (DRE)	DRE is the ratio of defects raised against the sum of post-production defects and the total number of defects raised.	Continuous
% Deferred Defects	Deferred defects is the ratio of the sum of open defects and deferred defects against the total number of defects.	Continuous
Reopen Defects	Reopen defects is the count of defects reopened after retesting.	Continuous

Metric	*Definition*	*Frequency*
Test Case Execution (TCE)	The percentage ratio of test cases executed vs. test cases planned.	Continuous
Test Automation Coverage	Total test cases actually automated (for the application) divided by the total test cases feasible to automate.	Continuous
Execution Cycle Time in Continuous Validation	Time taken to execute all the scripts in a particular test phase of continuous validation.	Continuous

Formatted: Font: Italic

Formatted Table

These metrics provide valuable insights into the quality and efficiency of the testing process, helping to identify areas for improvement and ensuring that the software meets the required standards before moving to the next phase.

Would you like more details on any specific metric or how to implement these in your dashboard?

Formatted: Font: Times New Roman, 14 pt

~~12.~~6. Reference(s)

1. Continuous Validation SharePoint: http://icp.ad.infosys.com/project/OFIVSCHZ/ContinuousValidation/SitePages/Home.aspx
2. Overall DevOps Strategy: Link to be updated
3. User Story document: Link to be updated
4. Gap Analysis: Link to be updated
5. Application User Stories: Link to be updated

Formatted: Font: (Default) Times New Roman, 14 pt, No underline, Font color: Auto

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: (Default) Times New Roman, 14 pt